PC System-on-a-Chip

MachZ

Training Book

Version 0.752

May 9, 2000

Legal Notice

THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN IS PROVIDED “AS-IS” AND WITHOUT A WARRANTY OF ANY KIND. YOU, THE USER, ACCEPT FULL RESPONSIBILITY FOR PROPER USE OF THE MATERIAL. ZF LINUX DEVICES, INC. MAKES NO REPRESENTATIONS OR WARRANTIES THAT THIS DATA BOOK OR THE INFORMATION CONTAINED THERE-IN IS ERROR FREE OR THAT THE USE THEREOF WILL NOT INFRINGE ANY PATENTS, COPYRIGHT OR TRADEMARKS OF THIRD PARTIES. ZF LINUX DEVICES, INC. EXPLICITLY ASSUMES NO LIABILITY FOR ANY DAMAGES WHATSOEVER RELATING TO ITS USE.

LIFE SUPPORT POLICY

ZF LINUX DEVICES' PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZF LINUX DEVICES, INC.

As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

MachZ, FailSafe FailSafe Boot ROM, Z-tag ZF-Logic, InternetSafe, OEMmodule SCC, ZF SystemCard, ZF FlashDisk-SC, netDisplay, ZF 104Card, ZF SlotCard, and ZF Linux Devices logo are trademarks of ZF Linux Devices, Inc. Other brands and product names are trademarks of their respective owners.

CAUTION: This is a Preliminary Specification. This specification should be used only for an architectural overview of the MachZ component.

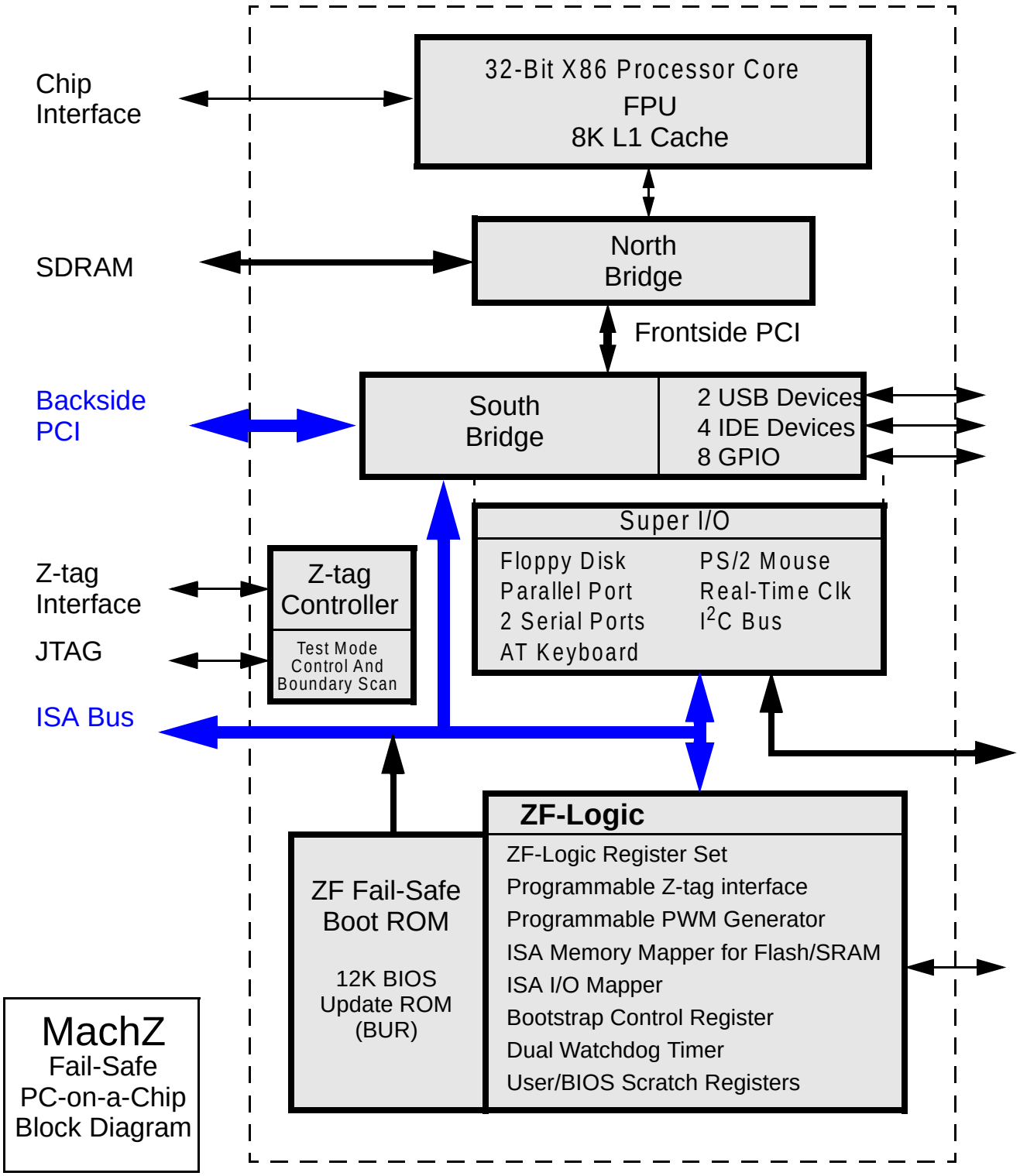
MachZ Training Book

Chapter 5 - ZF-Logic

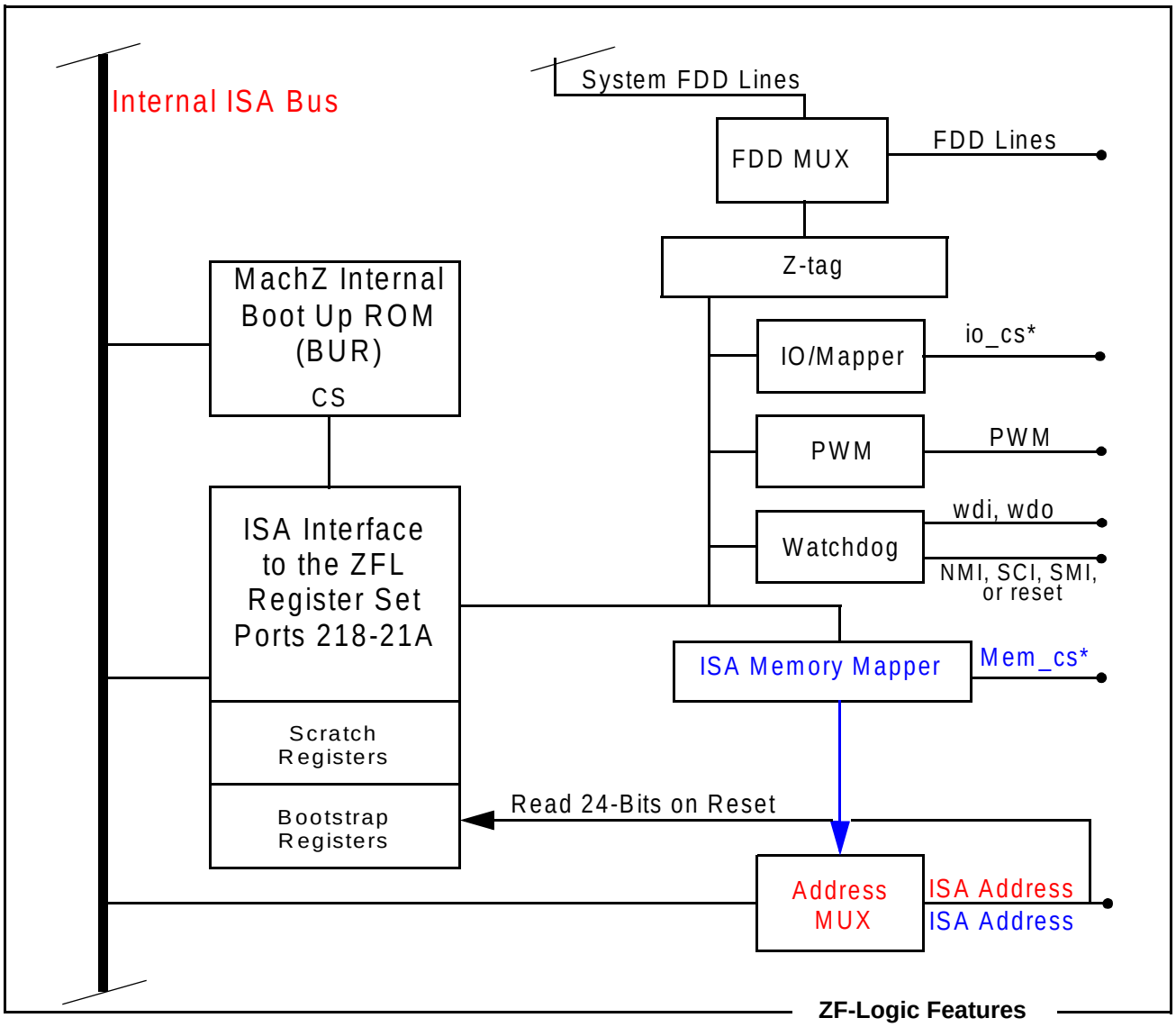
NOTE: See 'ZF-Logic and Clocking' on page 379 in the MachZ Data Book

NOTE References compatible with MachZ Developers Data Book version 0.752.

6 MachZ Block Diagram



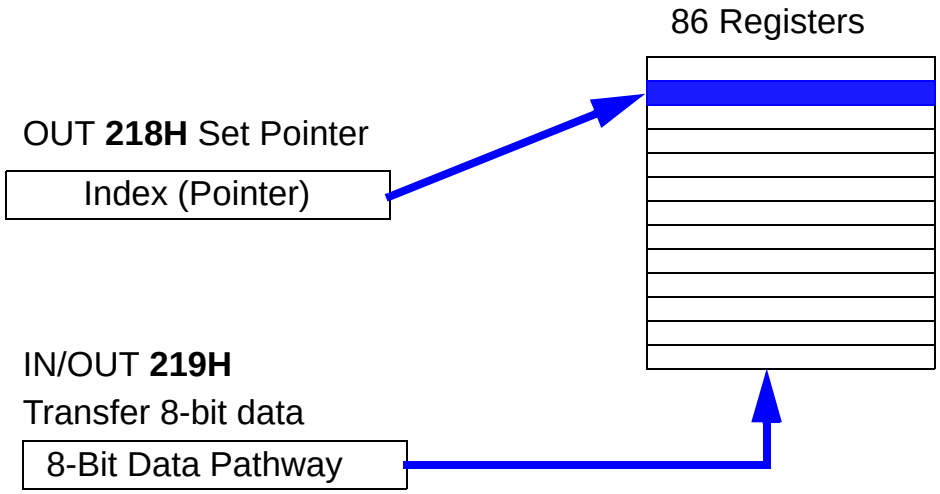
7 ZF-Logic Block Diagram



Note: The features of the ZFL include:

- ZFL Register Set in ISA I/O Space
- Programmable PWM generator
- Programmable Watchdog timer
- ISA Memory Mapper for Flash/SRAM
- ISA I/O Mapper General Purpose Chip Select (GPCS)
- Programmable Z-tag Interface
- Bootstrap Register (DIP switches/Pull-Ups) External Control of Boot Process
- User and BIOS Scratch Registers

8 ZF-Logic Register Space Access (8-Bit)



Index	8-Bit Data at Index	8-Bit Data at Index + 1	
02	ZF-Logic Revision (LSB) -- (02H)	ZF- Logic Revision (MSB) -- (03H)	

- ZF Logic is Controlled or Accessed through about 86 8-bit registers.
- Four ISA I/O Addresses are used provide Access to The Registers

QUESTIONS:

1] Why do we have 86+ Registers in the ZF Logic?

a] _____

_____

2] Why do we use 3 (rather than 86+) ISA Addresses?

a] _____

_____

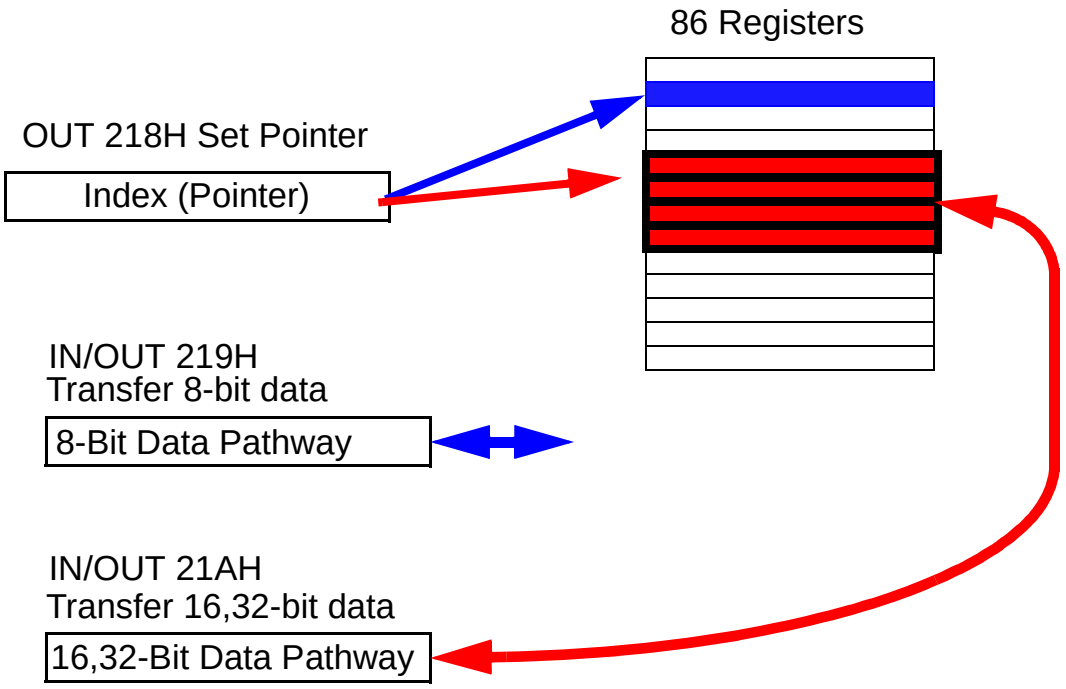
3] PROGRAMMERS: Write Code to Read Revision into CX register (asm) or 16-bit unsigned int (C)

a] _____

_____

[ ] Check Your Answers on page 39.

9 ZF-Logic Register Space Access (16, 32-Bit)



Index	8-Bit Data at Index	8-Bit Data at Index + 1	
02	**ZF-Logic Revision (LSB) -- (02H)**	**ZF- Logic Revision (MSB) -- (03H)**	

- ZF Logic is Controlled or Accessed through about 86 8-bit registers.
- Many of these registers are logically 16 or 32-bits wide
- By setting 218H to put to a 16 or 32-bit base, 2 or 4 consecutive bytes from the ZF register space can be transferred with a single IN or OUT (or C inp or outp) instruction.

Example: Read the 16-Bit Data at Index 02 to pick up the ZF-Logic Revision:

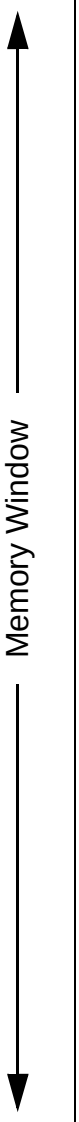
```
mov   al,02h     ; Index
mov   dx,218h    ; Index Address
out   dx,al      ; Set Index
                 ; read the value
mov   dx,21Ah    ; Data Viewport
in    ax,dx      ; AX=1234H (current revision of ZF-Logic)
```

10 ZF-Logic Registers

Index	8-Bit Data at Index	8-Bit Data at Index + 1	
02	ZF-Logic Revision (LSB) -- (02H)	ZF- Logic Revision (MSB) -- (03H)	
04	PWM Prescaler Low Byte -- (04H)	PWM Prescaler High Byte - (05H)	PWM
06	PWM duty cycle -- (06H)		
08	PWM I/O Control -- (08H)		
0A	PWM Read Output -- (0AH)		
0C	Watchdog 1 Count Low Byte -- (0CH)	Watchdog 1 Count High Byte -- (0DH)	W/D
0E	Watchdog 2 Count Value -- (0EH)	Watchdog Reset Pulse Length -- (0FH)	
10	Watchdog Control Low -- (10H)	Watchdog Control High -- (11H)	
12	Watchdog Events -- (12H)		
14	I/O Window 0 Base Low (14H)	I/O Window 0 Base High (15H)	I/O Window
16	I/O Window 0 Control (16H)		
18	I/O Window 1 Base Low (18H)	I/O Window 1 Base High (19H)	
1A	I/O Window 1 Control (1AH)		
1C	I/O Window 2 Base Low (1CH)	I/O Window 2 Base High (1DH)	
1E	I/O Window 2 Control (1EH)		
20	I/O Window 3 Base Low (20H)	I/O Window 3 Base High (21H)	
22	I/O Window 3 Control (22EH)		
24			

11 ZF-Logic Registers (2/3)

26	Memory Window 0 Base Bits 7-0	MW0 Base 15-12	MW0 Base 11-8
28	Memory Window 0 Base Bits 23-16	Memory Window 0 Base Bits 31-24	
2A	Memory Window 0 Size Bits 7-0	MW0 Size 15-12	MW0 Size 11-8
2C	Memory Window 0 Size Bits 23-16	Memory Window 0 Size Bits 31-24	
2E	Memory Window 0 Page Bits 7-0	MW0 Page 15-12	MW0 Page 11-8
30	Memory WIndow 0 Page Bits 23-16	Memory Window 0 Page Bits 31-24	
32	Memory Window 1 Base Bits 7-0	MW1 Base 15-12	MW1 Base 11-8
34	Memory Window 1 Base Bits 23-16	Memory Window 1 Base Bits 31-24	
36	Memory Window 1 Size Bits 7-0	MW1 Size 15-12	MW1 Size 11-8
38	Memory Window 1 Size Bits 23-16	Memory Window 1 Size Bits 31-24	
3A	Memory Window 1 Page Bits 7-0	MW1 Page 15-12	MW1 Page 11-8
3C	Memory Window 1 Page Bits 23-16	Memory Window 1 Page Bits 31-24	
3E	Memory Window 2 Base Bits 7-0	MW2 Base 15-12	MW2 Base 11-8
40	Memory Window 2 Base Bits 23-16	Memory Window 2 Base Bits 31-24	
42	Memory Window 2 Size Bits 7-0	MW2 Size 15-12	MW2 Size 11-8
44	Memory Window 2 Size Bits 23-16	Memory Window 2 Size Bits 31-24	
46	Memory Window 2 Page Bits 7-0	MW2 Page 15-12	MW2 Page 11-8
48	Memory Window 2 Page Bits 23-16	Memory Window 2 Page Bits 31-24	
4A	Memory Window 3 Base Bits 7-0	MW3 Base 15-12	MW3 Base 11-8
4C	Memory Window 3 Base Bits 23-16	Memory Window 3 Base Bits 31-24	
4E	Memory Window 3 Size Bits 7-0	MW3 Size 15-12	MW3 Size 11-8
50	Memory Window 3 Size Bits 23-16	Memory Window 3 Size Bits 31-24	
52	Memory Window 3 Page Bits 7-0	MW3 Page 15-12	MW3 Page 11-8
54	Memory Window 3 Page Bits 23-16	Memory Window 3 Page Bits 31-24	



12 ZF-Logic Registers (3/3)

56		BUR Base Low (57H)	
58	BUR Base High (58H)		
5A	Memory Control Low (5AH)	Memory Control Low (5BH)	
5C			
5E	Z-tag Data Write Register (5EH)		
60	Z-tag Data Read Register (60H)		
62	Bootstrap Bits 7-0 (62H)	Bootstrap Bits 15-8 (63H)	
64	Bootstrap Bits 23-16 (64H)		
66	I/O+Memory Window Map Events 66H		
68	Scratch Register 0 Low (68H)	Scratch Register 0 High (69H)	Scratch Registers
6A	Scratch Register 1 Low (6AH)	Scratch Register 1 High (6BH)	
6C	Scratch Register 2 Low (6CH)	Scratch Register 2 High (6CH)	
6E	Scratch Register 3 Low (6EH)	Scratch Register 3 High (6FH)	
70	Scratch Register 4 Low (70H)	Scratch Register 4 High (70H)	
72	Scratch Register 5 Low (72H)	Scratch Register 5 High (73H)	
74	Scratch Register 6 Low (74H)	Scratch Register 6 High (75H)	
76	Scratch Register 7 Low (76H)	Scratch Register 7 High (77H)	
78	Scratch Register 8 Low (78H)	Scratch Register 8 High (79H)	
7A	Scratch Register 9 Low (7AH)	Scratch Register 9 High (7BH)	
7C	Z-tag control register (7CH)	Z-tag Sequencer Divisor Register (7DH)	
7E	Z-tag Sequencer Waveform (7EH)	Z-tag Sequencer Strobe Points (7FH)	
80	Z-tag Sequencer Data (80H)	Z-tag Sequencer Status (81H)	

13 ISA Memory Windows for Flash / SRAM

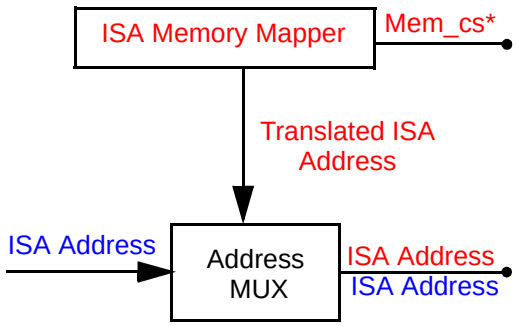
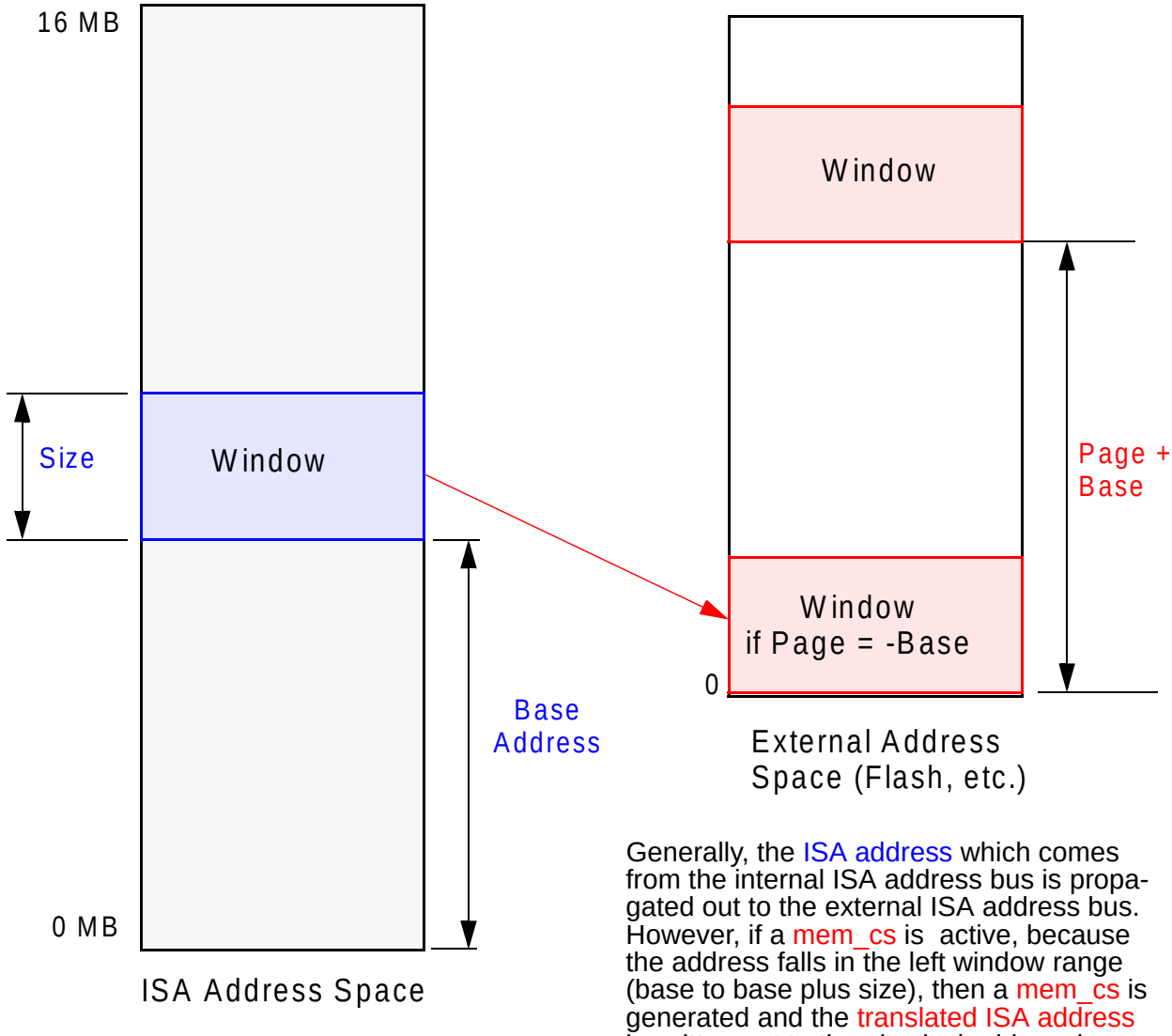


Table 0.1 Memory Mapper Pins

PKG	Name	Description
B04	Mem_cs0	ZF-Logic Memory Mapper CS 0
D05	Mem_cs1	ZF-Logic Memory Mapper CS 1
A03	Mem_cs2	ZF-Logic Memory Mapper CS 2
C04	Mem_cs3	ZF-Logic Memory Mapper CS 3



Generally, the ISA address which comes from the internal ISA address bus is propagated out to the external ISA address bus. However, if a mem_cs is active, because the address falls in the left window range (base to base plus size), then a mem_cs is generated and the translated ISA address is written out to the physical address bus.

14 Benefits of Memory Window Mapping

- The ZFL allows the MachZ to control up to four external memory devices on the ISA bus. These devices can be mapped into the system memory address space. Typically, this feature is used to map external Flash memory into the address space without external address decoding logic.
- Each device can occupy up to 16 Megabytes (occupying all 24 ISA address lines).
- In DOS mode these windows can be up to 256K bytes and reside only in upper 1 Mbyte DOS ROM area (C0000-FFFFF).
- In protected mode the windows can occupy all 24 ISA address lines (000000 - FFFFFF). This area is accessed in protected mode through memory space above system SDRAM. If the address is not in the system memory and no PCI device claims it then it is forwarded to the ISA bus. This makes the ISA bus useful multiple times in the upper memory area.
- Memory can be 8 or 16 bits wide, and may be write protected.

Memory Control Low -- Index 5AH

Bit	7	6	5	4	3	2	1	0
Function	w3_ro	w2_ro	w1_ro	w0_ro	w3_8	w2_8	w1_8	w0_8
Default	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Name	Function
7:4	wn_ro	Window n Read-write Control 0: Window N Is Read-write 1: Window N Is Read-only
3:0	wn_8	Window n Data Bus Width 0: Window N Uses 8-bit Data Access 1: Window N Uses 16-bit Data Access

15 Safety Aspects of Memory Windows

- Programmer/OS can request interrupts on Memory Window Overlap, Memory Change.
- Memory can be 8 or 16 bits wide, and may be write protected.
- Design Idea: Part of a Flash can be Write Protected by using two mem_cs* signals - one set R/W and one set R/O.

note: in case of overlapping addresses, mem_cs will *not* be asserted. The effect of the Events register is to cause notification via interrupt of this problem.

I/O and Memory Window Mapper Events -- Index 66H

Bit	7	6	5	4	3	2	1	0
Function	Reserved		Event Type		Memory Overlap	I/O Overlap	Memory Window Change	I/O Window Change
Default	0		0		0	0	0	0
R/W	R/O		R/W		R/W	R/W	R/W	R/W

Bit	Name	Function
7:6	Reserved	
5:4	Event Type	Generated event type 00 - No event 01- SCI 10 - NMI 11 - SMI
3	Memory Overlap	Enable resolve event on memory overlap 0: Disable event on memory overlap 1: Enable event on memory overlap
2	I/O Overlap	Enable event on I/O window overlap 0: Disable event on I/O window overlap 1: Enable event on I/O window overlap
1	Memory Access	Enable event on memory window change 0: disable event 1: enable event
0	I/O Access	Enable event on I/O window change 0: disable event 1: enable event

16 Memory Window Registers

1. First window settings
   - 2CH-2BH: Mem_cs0 window size — 0 - FFF000 = 16 MB / 0FFFFH
   - 30H-2FH: Mem_cs0 page — 0 - FFF000 = 16 MB / 0
   - 28H-27H: Mem_cs0 base address — 0 - FFF000 = 16 MB / F0000H

2. Second window settings
   - 38H-37H: Mem_cs1 window size
   - 3CH-3BH: Mem_cs1 page
   - 34H-33H: Mem_cs1 base address

3. Third window settings
   - 44H-43H: Mem_cs2 window size
   - 48H-47H: Mem_cs2 page
   - 40H-3FH: Mem_cs2 base address

4. Fourth window settings
   - 50H-4FH: Mem_cs3 window size
   - 54H-53H: Mem_cs3 page
   - 4CH-4BH: Mem_cs3 base address

5. 5BH-5AH: Control (R/W, 8/16 Width)

6. 66H: Events (SMI, etc.)

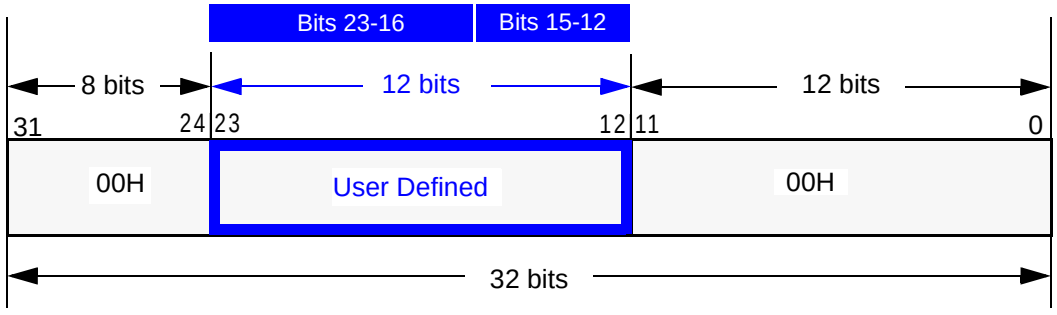
Initialization
Special Use of Mem_cs0

All the windows settings are initialized to 0 on power up reset. Setting a window size to 0 disables the window (the mem_cs*).

mem_cs0 is generally used for the boot ROM (flash). Thus the first window reset defaults are as shown above.

Window Size, Base Address, Page (translated address) all have a range of 0-16 MB. The mapping is disabled when *window size* is

There are two 8-bit registers each for Window Size, Base and Page. In each case, the two 8-bit registers supply bits 15-12 and bits 23-16 respectively. You may access these registers as a 32-bit operand. For example, to reference Window 0 Page Size, do a 32-bit transfer with the index set to **2EH** (covering 2E-2F-30-31 hex).



Fields in 32-bit memory settings registers

notes: SMI (System Management Interrupt) is used for legacy SMM BIOS; SCI (System Control Interrupt) is used for ACPI OS. Also, when determing whether or not an addresses emitted by the processor should generate a mem_cs*, the upper 8-bits of the 32-bit memory address are ignored. Thus a memory page can appear (alias) many times in the (CPU's) 4 GB address space.

17 ZF-Logic Memory Windows Review Ques-

4] Looking at the 'ZF-Logic Block Diagram' on page 7, what is the difference between the internal ISA bus and the output (on the lower right) called "ISA Address"? What is the function of the address multiplexer shown in the diagram, and when is the operative?

a] _____

_____

5] What are the necessary and sufficient conditions to cause a mem_CS signal to be asserted by the MachZ? Even if these conditions are met, when will a mem_cs signal not be asserted?

a] _____

_____

6] How many 32-bit ZF-Logic "memory window" registers are associated with the four mem_cs signals? See 'ZF-Logic Registers (3/3)' on page 12.

a] _____

_____

7] List the benefits to the customer which accrue from the MachZ memory window mapping feature.

a] _____

_____

8] What is the benefit to the customer due to the fact that his operating system may be notified based on memory window change? How this notification occur? See "I/O and memory window mapper events -- index 66 hex" on page 14.

a] _____

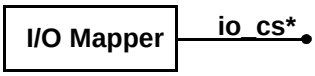
_____

9] What a special about mem_cs0? In order for that to work, what must happen?

a] _____

_____

[ ] Check Your Answers on Page page 40.



GPCS Pins

PKG	Name	Description
B03	io_cs0	ZF-Logic I/O Mapper GPCS 0
A02	io_cs1	ZF-Logic I/O Mapper GPCS 1
A01	io_cs2	ZF-Logic I/O Mapper GPCS 2
C03	io_cs3	ZF-Logic I/O Mapper GPCS 3

- MachZ has four GPCS (General Purpose Chip Select) signals mapped to io_cs* pins.
- Each io_cs* signal is assigned an address (or a set of consecutive addresses) in the ISA I/O space. This address, or set of consectutive addresses, is called the "window".
- Benefits: Each I/O Chip, requiring 1 to 16 consecutive I/O addresses, can be connected to the MachZ without glue logic.
- You can specify read-only or read-write, and 8 or 16-bit wide data transfers.
- Benefit: if your I/O chip is 8 or 16-bits wide, you do not need special logic to inform the MachZ -- it is set up in the control registers.
- There is no address translation here: just appropriate io_cs* generation.

notes: For example, if the chip you wish to connect to the MachZ using one of the io_cs pins has four ports (such as the old 8255 chip), you would want the chip select to be asserted for four consecutive addresses. The chip itself would differentiate between the addresses by looking at the low two bits of the ISA address bus

19 GPCS (I/O Mapper) Register Set

1. GPCS 0 settings

- 15H-14H: io_cs0 base address — 0000 - FFFFH (CPU I/O Range)
- 16H: io_cs0 control — RO/RW, 8/16 bit data, size, enable

2. GPCS 1 settings

- 19H-18H: io_cs0 base address
- 1AH: io_cs0 control

3. GPCS 2 settings

- 1DH-1CH: io_cs0 base address
- 1EH: io_cs0 control

4. GPCS 3 settings

- 21H-20H: io_cs0 base address
- 22H: io_cs0 control

5. Events Register (66H) — Window Change, Overlap

- A chip select may be generated for any 1 to 16 consecutive addresses in any part of the CPUs to 64K I/O address space. The Intel architecture supports only 64K of I/O address space, so the 16-bit base address provides precise and complete locating of the chip select decode.
- Sufficient control information is required on a per chip-select basis, to require one control byte for each I/O chip select.
- The same events register (66H) is used for the memory and I/O mapping. The same errors are detected.

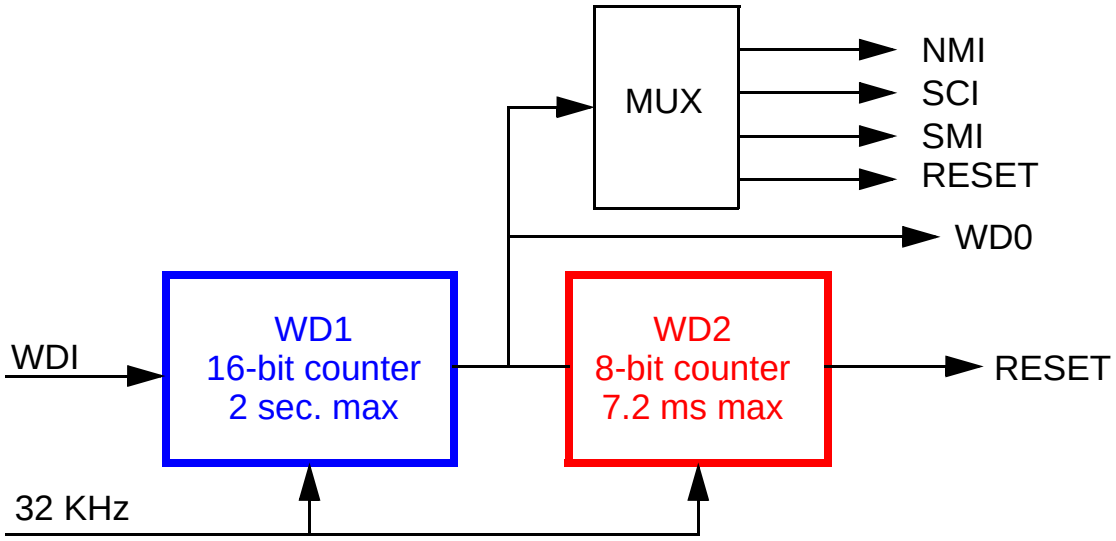
20 GPCS (I/O Mapper) Control Registers

I/O Window “N” Control

Bit	7	6	5	4	3	2	1	0
Func-tion	win_ro	16_bit	act_lvl	win_en	win_siz: I/O Window Size (1-16)			
Default	0	0	0	0	0 (Size is 1)			
R/W	R/W	R/W	R/W	R/W	R/W			

Bit	Name	Function
7	win_ro	I/O window read/write control 0: Access is read-write 1: Access is read-only Setting window to read-only mode disables IOW_N signal on ISA bus for IO window address range.
6	16_bit	I/O window datapath width 0: 8-bit wide access 1: 16-bit wide access
5	act_lvl	io_cs active level 0: io_cs is active low 1: io_cs is active high
4	win_en	I/O window enable in I/O space 0: I/O window is disabled 1: I/O window is enabled
3:0	win_siz	Number of consecutive 8-bit I/O addresses to decode starting from I/O window base. The number of consecutive addresses decoded is win_siz + 1. For example, setting the window size to 0 enables one I/O address at I/O window base. Setting size to 0Fh will enable I/O window of 16 addresses starting from I/O window base.

21 Watchdog Timer



- Whenever WD1 is not reloaded during a pre programmed interval it generates an event to notify the system of an error condition.
- The first watchdog timer is initialized to a 16-bit timeout value through registers 0Ch and 0Dh. After enabling through control register (10h) it starts the countdown to zero. The first watchdog timer can be reloaded to an initial value by writing into control register (10h) or asserting watchdog external control pin on MachZ (WDI).
- Whenever the first watchdog is not reloaded during the timeout value it generates an event to notify the system of an error condition and outputs the logical "1" to a watchdog output pin on MachZ (WDO). The notification event can be routed to NMI, SMI, SCI or it can reset the system immediately.
- The second watchdog timer 8-bit timeout value is initialized through register 0Eh and starts counting down after WD1 time-out. When the WD2 counter reaches zero, it will unconditionally cause system reset.

22 ZF-Logic Registers for the Watchdog Timer

0C	Watchdog 1 Count Low Byte (0CH)	Watchdog 1 Count High Byte (0DH)
0E	Watchdog 2 Count Value (0EH)	Watchdog Reset Pulse Length (0FH)
10	Watchdog Control Low (10H)	Watchdog Control High (11H)
12	Watchdog Status (12H)	

- Count Registers - Reload Values for Watchdog Timers
- Watchdog Reset Pulse Length - the number of 32kHz ticks to hold the system reset signal low
- Watchdog Control: Enable/Disable, and MUX Control

ZF-Logic Registers for the Watchdog Timer

Watchdog Control Low -- Index 10H

Bit	7	6	5	4	3	2	1	0
Func-tion	reserved		wd2 load	wd1 load	reserved		wd2 enable	wd1 enable
Default	0		0	0	0		**0**	**0**
R/W	R/O		R/W	R/W	R/O		R/W	R/W

Bit	Name	Function
7:6	Reserved	
5	wd2 load	Reload WD2 counter. Active event for this bit is transition from 0 to 1
4	wd1 load	Reload WD1 counter. Active event for this bit is transition from 0 to 1
3:2	Reserved	
1	wd2 enable	Enable wd2 0: WD2 is disabled 1: WD2 is enabled
0	wd1 enable	Enable wd1 0: WD1 is disabled 1: WD1 is enabled

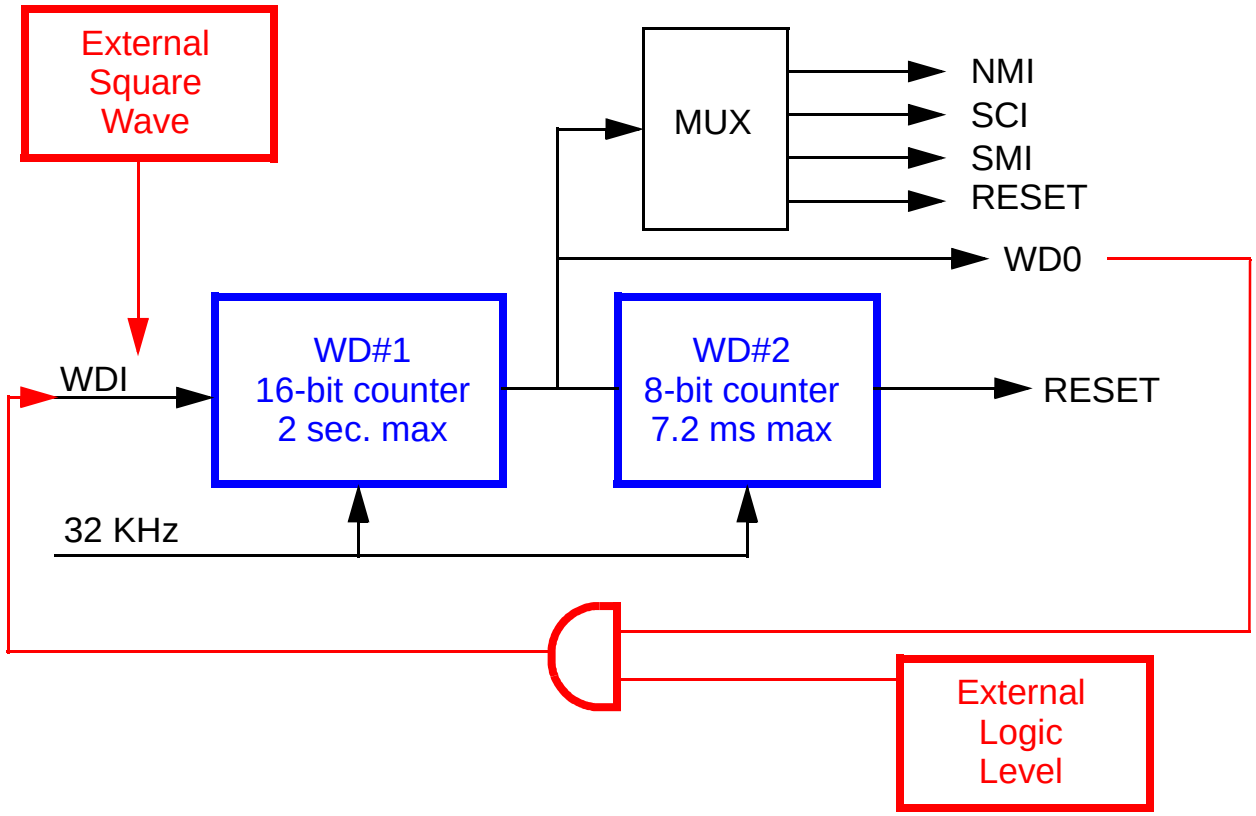
24 ZF-Logic Registers for the Watchdog Timer

Watchdog Control High -- Index 11H

Bit	7	6	5	4	3	2	1	0
Function	reserved	wdi_en	wdo_-1	wdi edge	wd1 reset	wd1 SMI	wd1 NMI	wd1 SCI
Default	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Name	Function
7	Reserved	
6	wdi_en	**Enable the assertion of WDI input pin on MachZ to to reload the watchdog 1 counter** 0: WDI input ignored 1: WDI assertion reloads watchdog 1 counter
5	wdo_-1	Create output on WDO output pin on MachZ at WD1 time-out or one 32kHz clock tick before 0: WDO signal will be set high on WD1 expiration 1: WDO signal is set high one clock tick before WD1 expires. WD1 events will always occur at WD1 time-out and are not affected by wdo_-1 bit setting. This feature permits **automatic reload** of WD1 when WDO is wired to WDI.
4	wdi edge	Active front of WDI input 0: WDI is asserted on 0->1 transition 1: WDI is asserted on 1->0 transition
3	wd1 reset	WD1 generates **system reset** on time-out 0: WD1 will not generate system reset on time-out 1: WD1 will generate system reset on time-out
2	wd1 SMI	WD1 generates **SMI** on time-out 0: wd1 will not generate SMI on time-out. 1: wd1 generates SMI on time-out
1	wd1 NMI	WD1 generates **NMI** on time-out 0: wd1 will not generate NMI on time-out 1: wd1 generates NMI on time-out
0	wd1 SCI	WD1 generates **SCI** on time-out 0: wd1 will not generate SCI on time-out 1: wd1 generates SCI on time-out

25 External Control of Watchdog Timeout



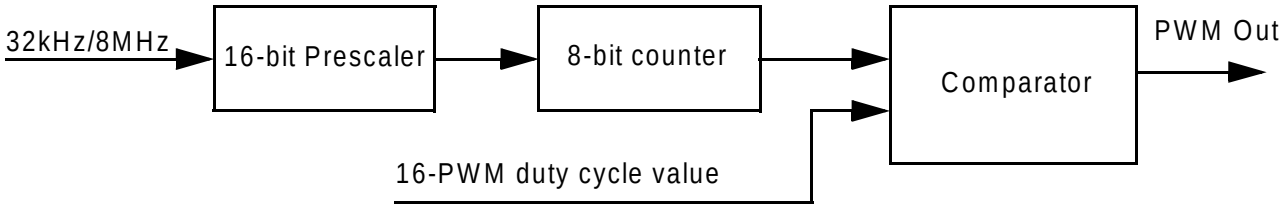
- Square Wave on WDI may reset (external reset)
- Wiring WD0 to WDI through Gate allows use of Logic Level rather then Square Wave
- When wiring WDO to WDI, to prevent event (NMI, etc) (so long as gate is on) then use wdo-1 to set reset of WD#1 1 event before expiration

Notes: There is a WDO and WDI pin -- the WDI can be programmed to reload WD#1. If you toggle WDI (falling or rising) you can prevent the WD from ever expiring. The benefit of this is that so long as an external square wave is coming in, the WD never expires. You are thus using an EXTERNAL way of keeping the MachZ from resetting -- you are watching for an external “dead man” switch.

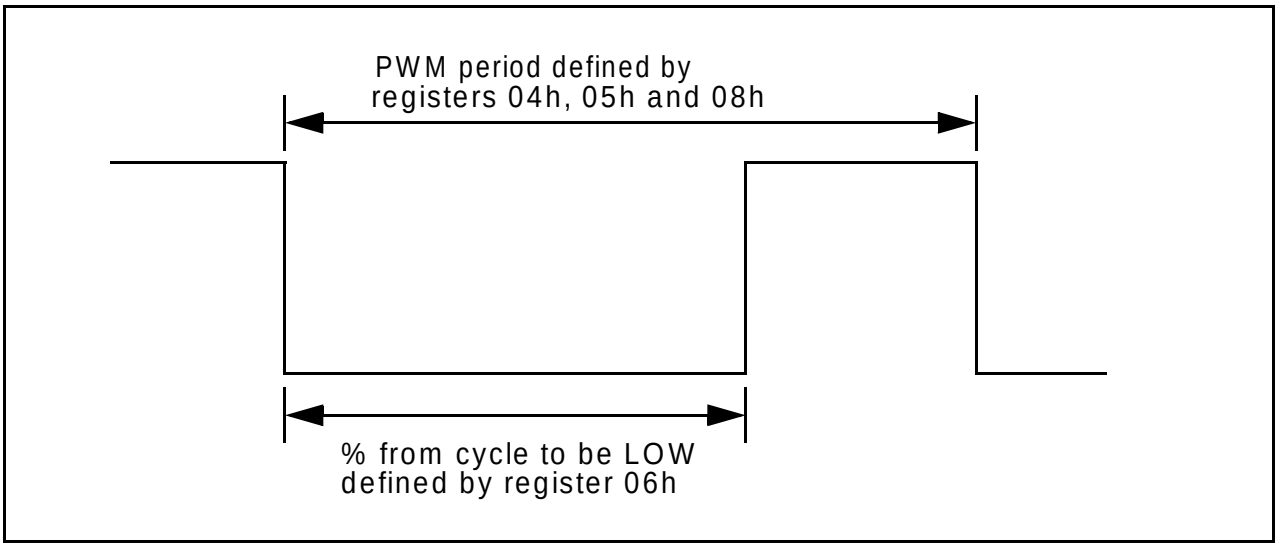
To do this, you need to have an outside event generator. Let’s assume that all you have is a logic signal which shows you if the external system is working or not. You can then connect WDO to WDI with an OR gate or AND gate to that external signal.

If you set it up this way, then you set wdo-1 to generate event 1 pulse before expiring. If you did not do this, it would expire. See wdo-1 in "Watchdog Control High -- Index 11H" on page 24

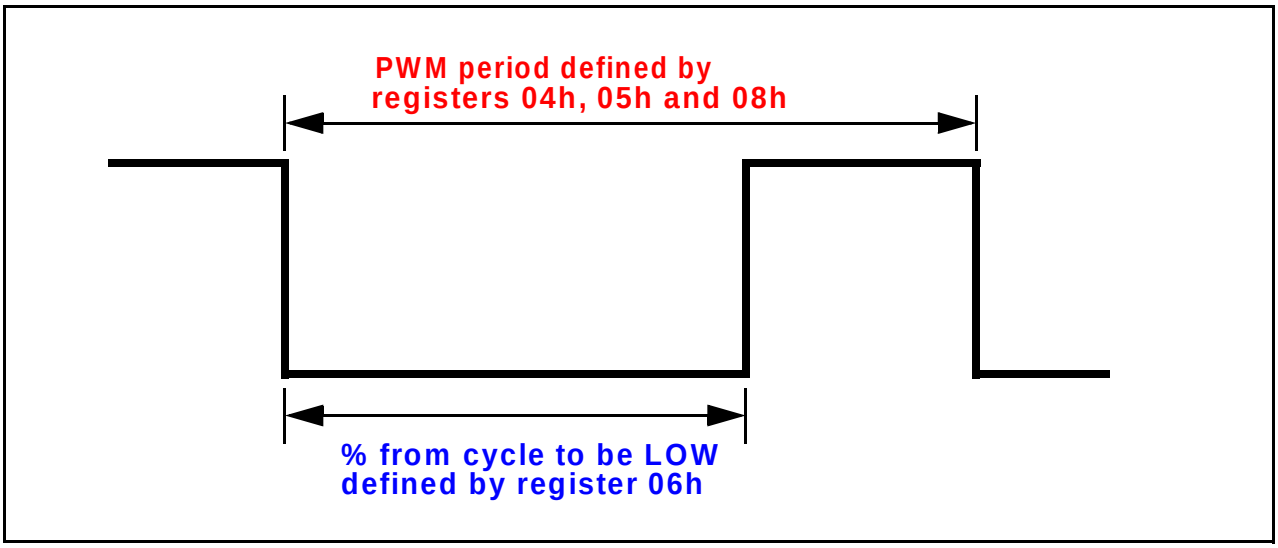
26 PWM Generator



- The PWM (Pulse Width Modulation) output may be used to create DC control voltage for an LCD backlight or any other device that requires this feature. The conversion is done by integrating variable duty cycle signal externally. At higher frequencies it may be used to control external transformer for DC/DC conversion.



27 PWM Generator Period / Duty Cycle



ZF-Logic Index for the PWM Generator

04	PWM Prescaler Low Byte -- (04H)	PWM Prescaler High Byte - (05H)
06	PWM duty cycle -- (06H)	
08	PWM I/O Control -- (08H)	
0A	PWM Read Output -- (0AH)	

- PWM Prescaler: Divides 8MHz or 32kHz input clock selected at PWM control register. Actual divisor is 16-bit PWM divisor word (combined of registers 04h and 05h) + 1
- PWM Duty Cycle: Sets the % of the cycle to be low. (0 = 100%, 255 = 0%).
- PWM I/O Control: Includes selection of 32kHz clock or 8 MHz ISA clock

28 PWM Generator - I/O Control Register

PWM I/O Control -- Index 08H

Bit	7	6	5	4	3	2	1	0
Func-tion	reserved		enable direct	direct output	reserved		slow-fast clksrc	Enable PWM
Default	0		0	0	0		0	0
R/W	R/O		R/W	R/W	R/O		R/W	R/W

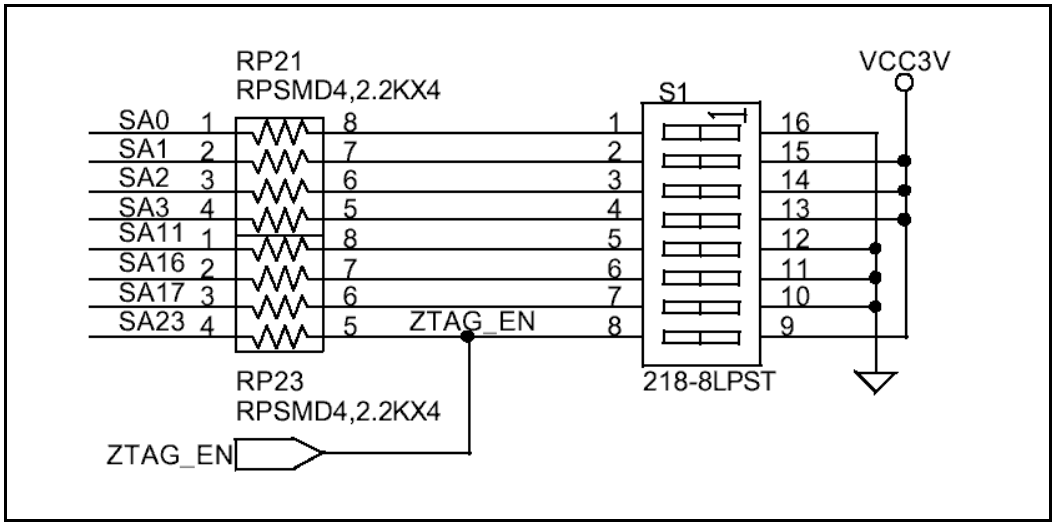
Bit	Name	Function
7:6	Reserved	
5	enable direct	**Enables direct control of PWM output by bit 4** 0: PWM drives the output 1: Bit 4 of register 08H drives the PWM output pin
4	**direct output**	The value of PWM output when bit of register 08h is set to 1
3:2	Reserved	
1	slow-fast (clksrc)	**Selects the PWM prescaler input clock** 1: PWM is clocked by 32kHz clock 0: PWM is clocked by 8 MHz ISA clock
0	Enable/Disable PWM	**Enable/Disable PWM** output 0: PWM is disabled 1: PWM is enabled

29 Boot Parameters Register

- When **power-on reset** is asserted 24 signals are read into the Boot Parameters Register (configuration register) from the ISA Address Bus.
- In a typical design, DIP switches or jumpers are used (with appropriate resistors) set to the bits in the BootStrap Register.

ZF-Logic Index for the Boot Parameters Register

62	Bootstrap Bits 7-0 (62H)	Bootstrap Bits 15-8 (63H)
64	Bootstrap Bits 23-16 (64H)	

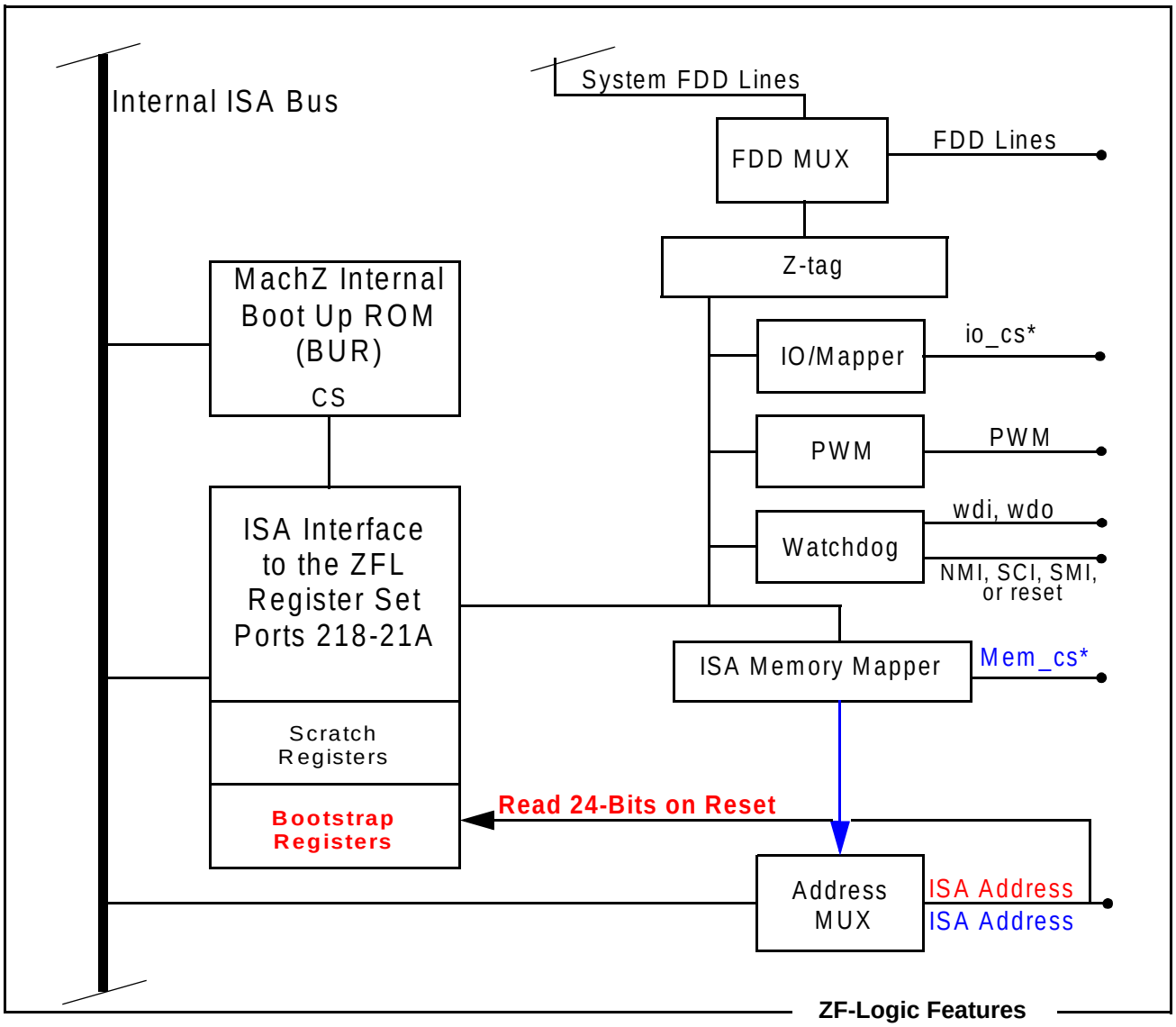


The ISA address bus (pins SA0-SA23) is tri-stated during the reset pulse. It contains on-chip weak (about 20K) pull-ups and pull-downs to set the default state of the bootstrap register. To override this, we use a 2.2K pull-up or pull down and a DIP switch or jumper. Once the reset pulse is done, the ISA bus has sufficient drive to overcome the effect of these 2.2K resistors.

Thus, conceptually the ISA address bus has three "modes": (1) the weak on-chip pull-ups/ pull-downs which are operative during the tri-state; (2) the 2.2K pull-ups/pull-downs which may be activated via DIP switches; and (3) the normal execution time mode where the drive of the ISA address bus will override these resistors.

Since the Boot Parameters Register is read only, the values sampled on the ISA bus on the trailing edge of reset are "permanent" until the next hardware reset. Software can read

30 Boot Parameters Register

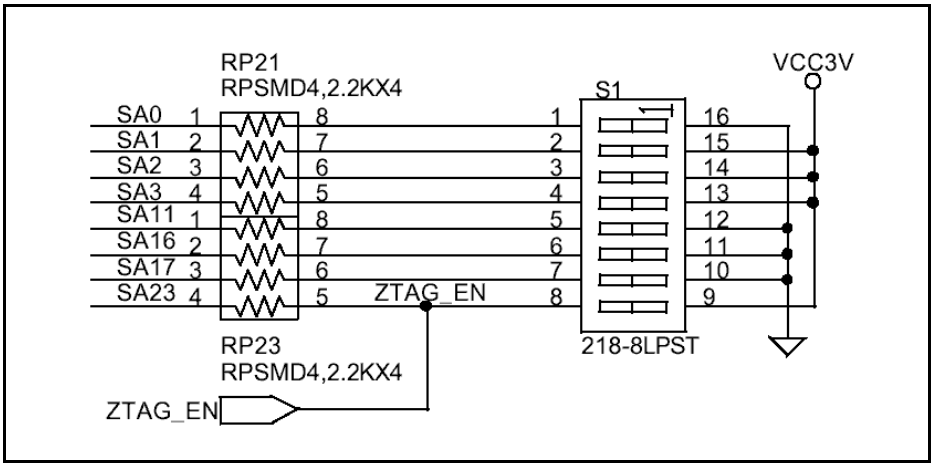


Example:

; read bootstrap registers as 32-bit value into EAX

```
0102   B0 62          mov   al,62h
0104   BA 0218        mov   dx,ZFLINDEX
0107   EE             out   dx,al
0108   BA 021A        in    eax,dx
010D   66| 25 00FFFFFF and   eax,0FFFFFFh
```

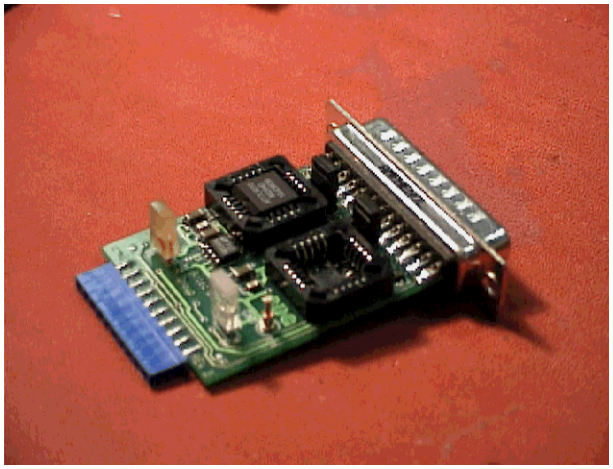
31 Boot Parameters Register



Sample DIP Switch Settings

SW	Function
1	ON - USER DEFINED
2	ON - USER DEFINED
3	ON - USER DEFINED
4	ON - USER DEFINED
5	ON - EXT ROM when Z-tag enabled[a]
6	System Clock Speed
7	System Clock Speed
8	ON - Boot from BUR

a. This bit should NOT be used in real designs. It is for testing only.



ISA BIT	Index	Bit	Name	Default	Function
23	64H	7	Boot from BUR (sometimes called ZTAG_EN)	0	Boot from BUR 1 = Boot from BUR 0 = Boot from Flash

When you plug the dongle in, it automatically sets BS23.

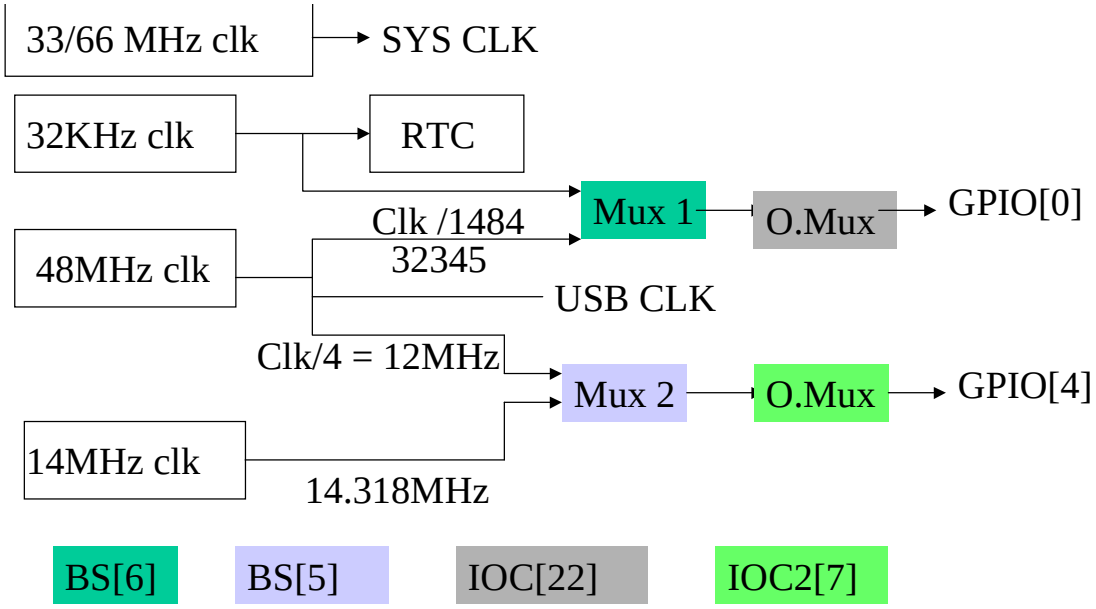
32 Boot Parameters Register

Composite BootStrap Register Map

ISA BIT	Index	Bit	Name	Def	Function
0-3	62H	0-3	User Defined	0	User Defined
5	62H	5	14 Mhz clock source	0	14MHz Clock Source If 1, derive from 48Mhz. If 0, use mhz14_c pin. [AF16]
6	62H	6	32 KHz	0	32KHz Clock Source If 1, derive for 48MHz. If 0, use 32KHZC [AF01]
9	63H	1	3rd PCI Request	0	Third PCI Request/Grant 1 = drq1 = req2_n and dack1_n = gnt2_n
11	63H	3	Reserved	1	Internal / External BUR Source. 0 = External BUR 1 = Internal BUR
12	63H	4	ISA Boot ROM Width	1	ISA Boot ROM Width 0 = 16 bit 1 = 8 bit
16 17	64H	0 1	486 Clk Multiply	11	00 - Sys Clk * 1 01 - Sys Clk * 2 11 - Sys Clk * 3 (default) 10 - Sys Clk * 4
18	64H	2	FPCI divide	0	Frontside PCI Clock Divide. 0- SysClk 1 - SysClk / 2...
19	64H	3	BPCI divide	0	Backside PCI Clock Divide.[a.] 0- SysClk 1 - SysClk / 2..
20	64H	4	BPCI Select	1	Backside PCI Clock Select.[a] 0 - External clock. 1- Internal clock.
23	64H	7	Z-tag enable	0	Causes BUR Boot. Enables the Z-Tag Interface and BUR if high.

a. If Bit 20 is 1, then Bit 19 has no affect.

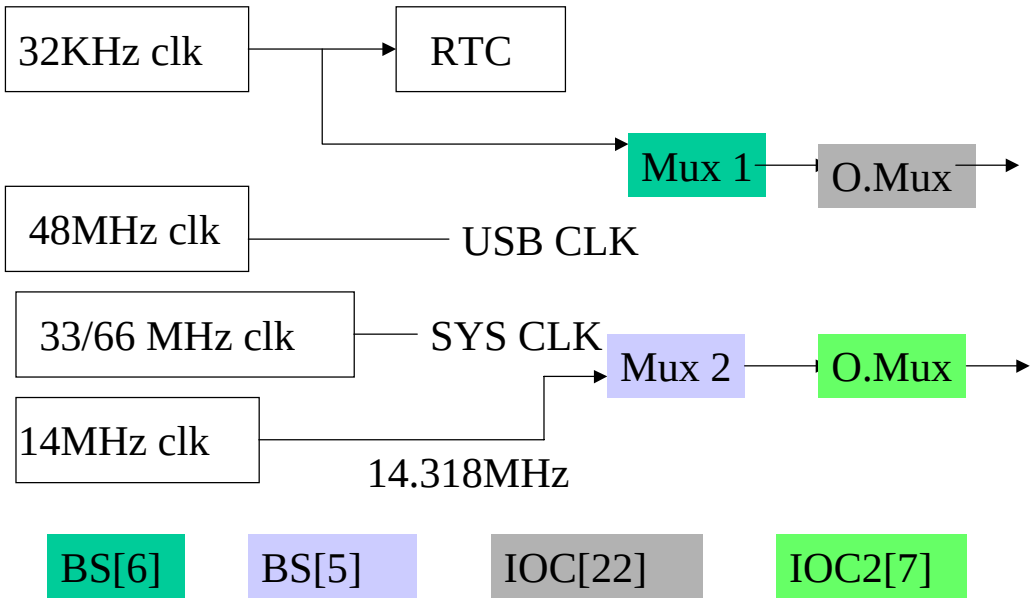
Full Clocking Diagram



ISA BIT	Index	Bit	Name	Def	Function
5	62H	5	14 Mhz clock source	0	14MHz Clock Source If 1, derive from 48Mhz. If 0, use mhz14_c pin. [AF16]
6	62H	6	32 KHz	0	32KHz Clock Source If 1, derive for 48MHz. If 0, use 32KHZC [AF01]

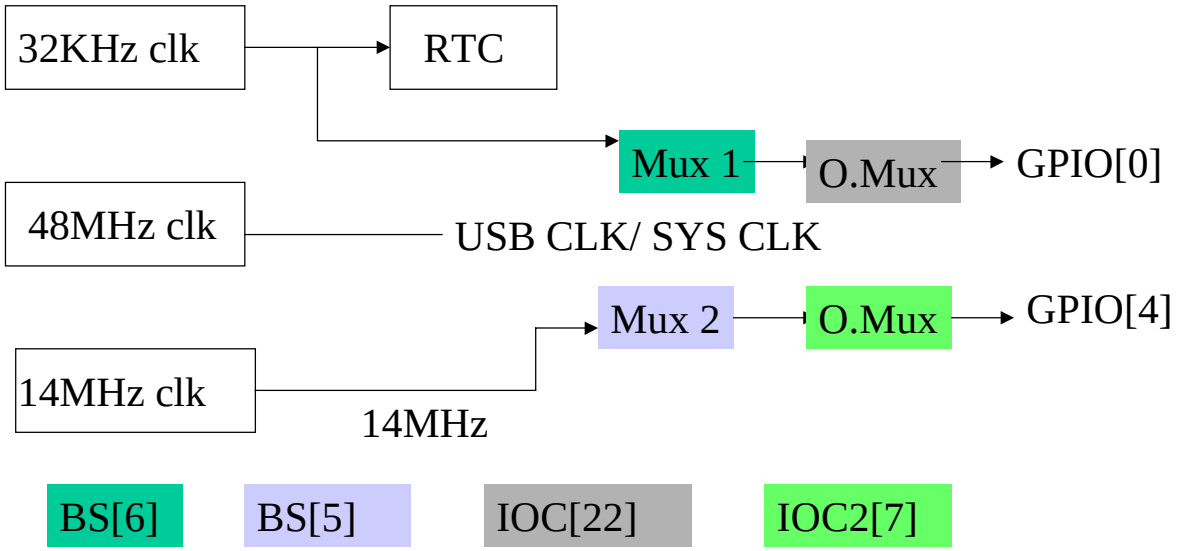
NOTES: The Mach Z System-on-a-Chip has various clocking options. These options represent different trade-offs that the designer must investigate to come to the best solution for the application being considered. Essentially, the chip can be clocked using as many as four sources or as few as one.

34 Clocking Choices (1)



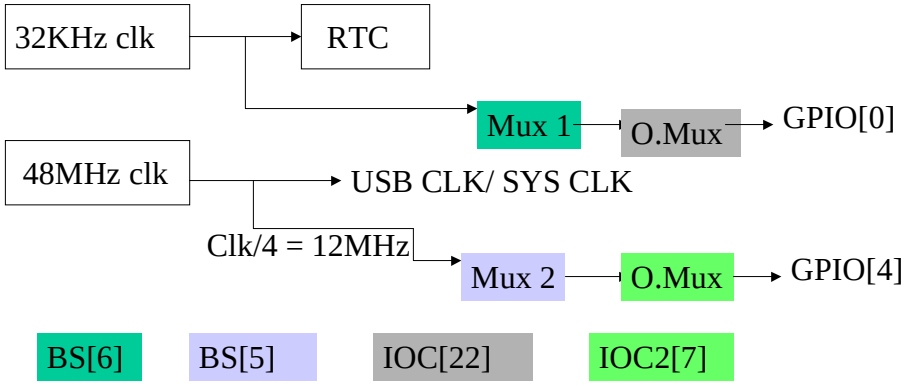
35 Clocking Choices (2)

• **LIMITED OPTIONS ON DRAM AND CPU CLOCK.**



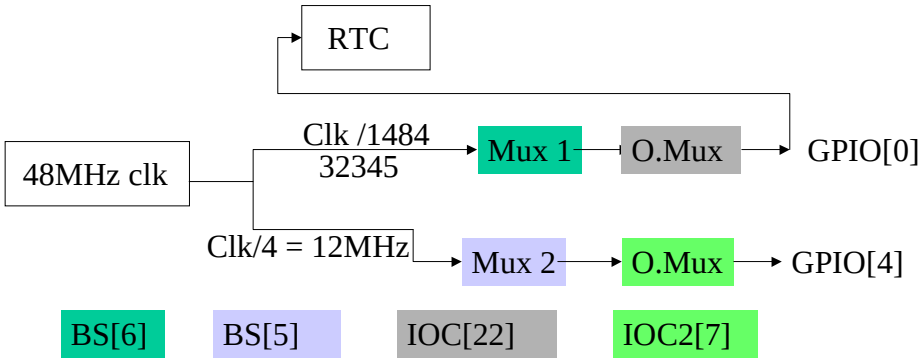
36 Clocking Choices (3)

• LIMITED OPTIONS ON DRAM AND CPU CLOCK, ISA CLOCKING OFF.



37 Clocking Choices (4)

- Simplest clock choice: Lose RTC on Power down, incur error in time, and ISA timing.



Performance

- What is the best clocking for the customer?

Application	Key Clock
Kiosc	SYSCLK @ 66, CPU 2X, PCI /2
Taxi	SYSCLK @ 48 CPU 2X, 32KHZ
POS	SYSCLK @ 48, CPU 2X, 32KHz
Agriculture	SYSCLK 48 ONLY

Answers to Questions

<page 8>

1] Why do we have 86+ Registers in the ZF Logic?

a] The actual address range of the ZF-Logic registers is index 2 through 81H, or about 128 byte addresses. Of these, approximately 86 off the byte addresses have read/write data fields. We need this many registers to contain all of the control and status bits for all of the functions built into the ZF-Logic.

2] Why do we use 3 (rather than 86+) ISA Addresses?

a] The industry standard architecture, a derivative of the first IBM personal computer, has many assigned and reserved addresses within the I/O address space. In order to have a minimum profile, we access all of the ZFL control logic registers using three and only three ISA I/O addresses.

3] PROGRAMMERS: Write Code to Read Revision into CX register (asm) or 16-bit unsigned int (C)

a]

```
mov   al,02h     ; Index
mov   dx,218h    ; Index Address
out   dx,al      ; Set Index
                 ; read the value
mov   dx,21Ah    ; Data Viewport
in    ax,dx      ; AX=1234H
```

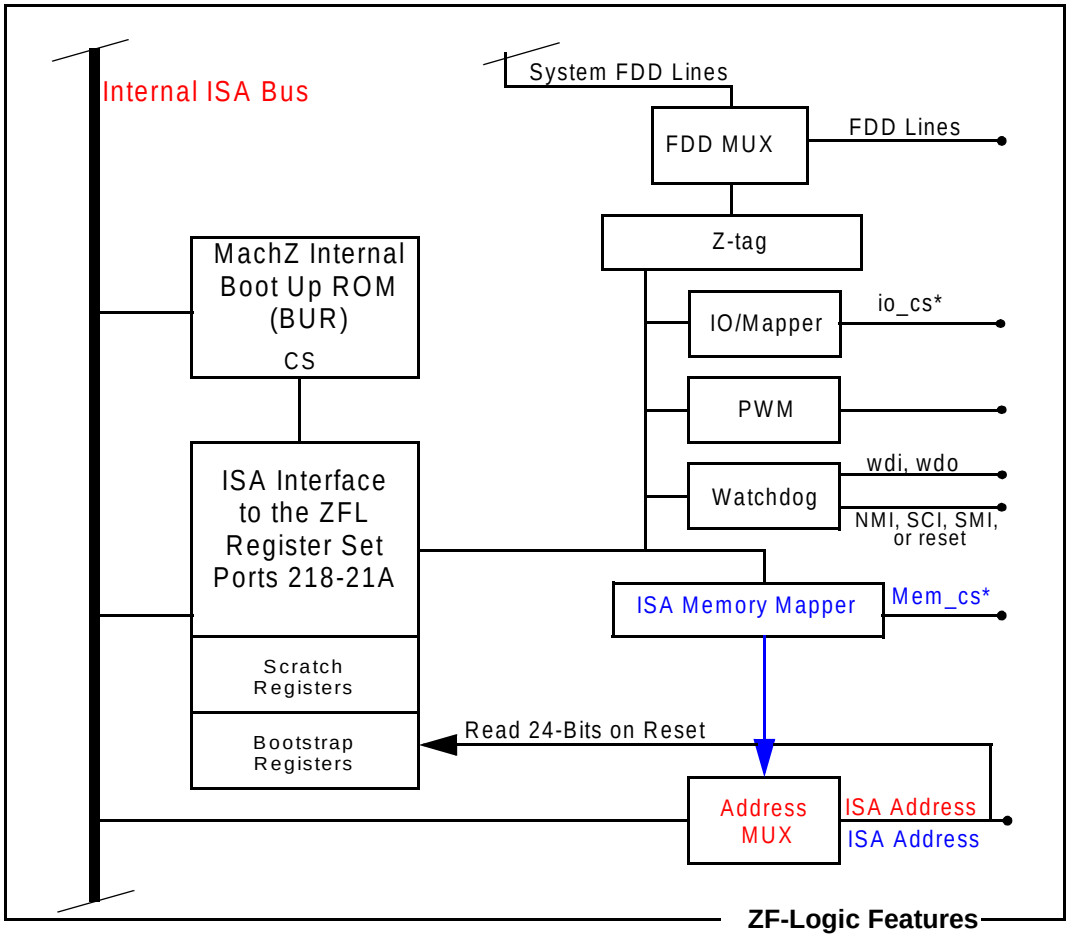
```
; this solution cheats and uses a 16-
bit transfer that we have not covered
yet.
```

```
uchar ucLow, ucHigh;
unsigned int uiRevision;
#define INDEX 0x218
#define TRANSFER 0x219
#define REVISION_LSB 2
#define REVISION_MSB 3
```

```
// solve using 8-bit transfers

    outp (INDEX, REVISION_LSB);
    ucLow = inp (TRANSFER);
    outp (INDEX, REVISION_MSB);
    ucHigh = inp (TRANSFER);
    uiRevision = ucHigh * 256 + ucLow;

// alternative using 16-bit transfer

#define TRANSFER1632 0x218

    outp (INDEX, REVISION_LSB);
    uiRevision = inpw (TRANSFER1632);
```

4] Looking at the 'ZF-Logic Block Diagram' on page 7, what is the difference between the internal ISA bus and the output (on the lower right) called "ISA Address"? What is the function of the address multiplexer shown in the diagram, and when is the operative?

a] Addresses on the internal ISA bus are generally routed to the output pins on the MachZ whenever there are memory read, memory write, input read, and output write, transfer cycles. That is, unless the ISA memory mapper grabs certain memory read and memory write cycles, the addresses from those cycles propagates directly out on the ISA address bus. However, if a memory address is in the range specified by the base to base + size, then two things happen: (1) the appropriate mem_cs is generated; and (2) the ISA memory mapper provides a translated address and that is the address which goes out on the external ISA address bus pins. Thus the function of the address multiplexer shown in the diagram is to route to the external ISA address bus either (1) the ISA address from the internal ISA bus, or (2) the translated ISA address from the ISA memory mapper.



5] What are the necessary and sufficient conditions to cause a mem_CS signal to be asserted by the MachZ? Even if these conditions are met, when will a mem_cs signal not be asserted?

a] In order to get one of the four mem_cs signals to be generated, a memory read or memory write generated by the CPU must (1) reach the internal ISA bus, and (2) be between the window base address and the window base+size.

Note that mem_cs signal not be asserted in the window size equals 0 (as it turns off ISA memory mapping for that window), or if this memory window overlaps another memory window. Review 'ISA Memory Windows for Flash / SRAM' on page 13.

6] How many 32-bit ZF-Logic "memory window" registers are associated with the four mem_cs signals? See 'ZF-Logic Registers (3/3)' on page 12.

a] There are 3 32-bit registers per memory window or a total of 12 32-bit registers. These registers contained the base, size, and target offset (page). There are other control registers (a control and an event register), but the basic window mapping occurs using the 3 32-bit registers per window.

7] List the benefits to the customer which accrue from the MachZ memory window mapping feature.

a] The first, and most important benefit, is that no extra hardware or glue logic is required to in order to manage the chip selects for up to 4 SRAM or flash devices. In addition, each device may be programmed to be read-write or read-only. Further, without any extra external logic, a data bus width for each device may be specified to be 8 or 16 bits. Besides that, there is an event register which enables various interrupts to be received by the operating system based on misuse of the memory mapping feature.

8] What is the benefit to the customer due to the fact that his operating system may be notified based on memory window change? How this notification occur? See 'I/O and Memory Window Mapper Events -- Index 66H' on page 15.

a] If an unauthorized program purposely or accidentally attempts to change the size of a memory window it can represent a serious bug causing product release and development delays (during development) or subsequent failures in the field. Well behaved hardware will notify the the operating system via interrupts if something happens which is not supposed happen. The ability of the MachZ to notify the operating system via interrupts if someone tries to change the memory window parameters, or if there is an inadvertent overlap of the memory windows, enhances system integrity.

9] What is special about mem_cs0? In order for that to work, what must happen?

a] When the CPU powers up, it does an instruction fetch from 000FFFF0H, which in all Intel x86 computers is routed to the last 16 bytes of the boot ROM. In MachZ designs, we connect mem_cs0 to the flash chip containing the boot code. The MachZ chip, on power-up reset, initializes 3 32-bit registers for mem_cs0 such that this instruction fetch will read from a flash device.

As a technical note: The window size is set to 64 K, and the base address to F0000 (the last 64K of the 1MB of "real" memory), and the page to 0. This means that instruction fetches in the top 64K of the 1MB real address space will read from the first 64K of the flash chip.

1. First window settings

- 2CH-2BH: Mem_cs0 window size 0 - FFF000 = 16 MB / 0FFFFH
- 30H-2FH: Mem_cs0 page 0 - FFF000 = 16 MB / 0
- 28H-27H: Mem_cs0 base address 0 - FFF000 = 16 MB / F0000H

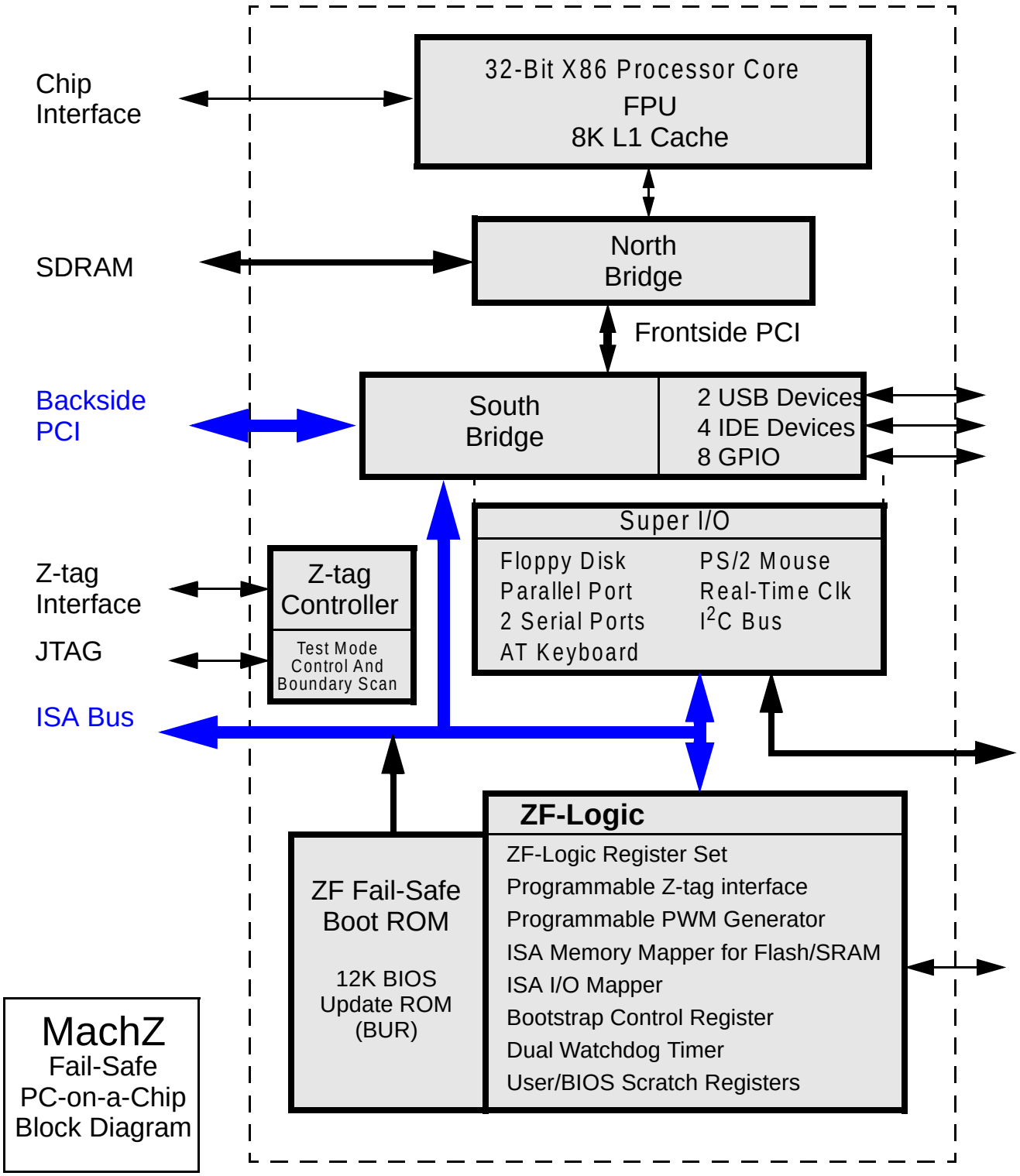
MachZ Training Book

Chapter 6 - Z-Tag / BUR

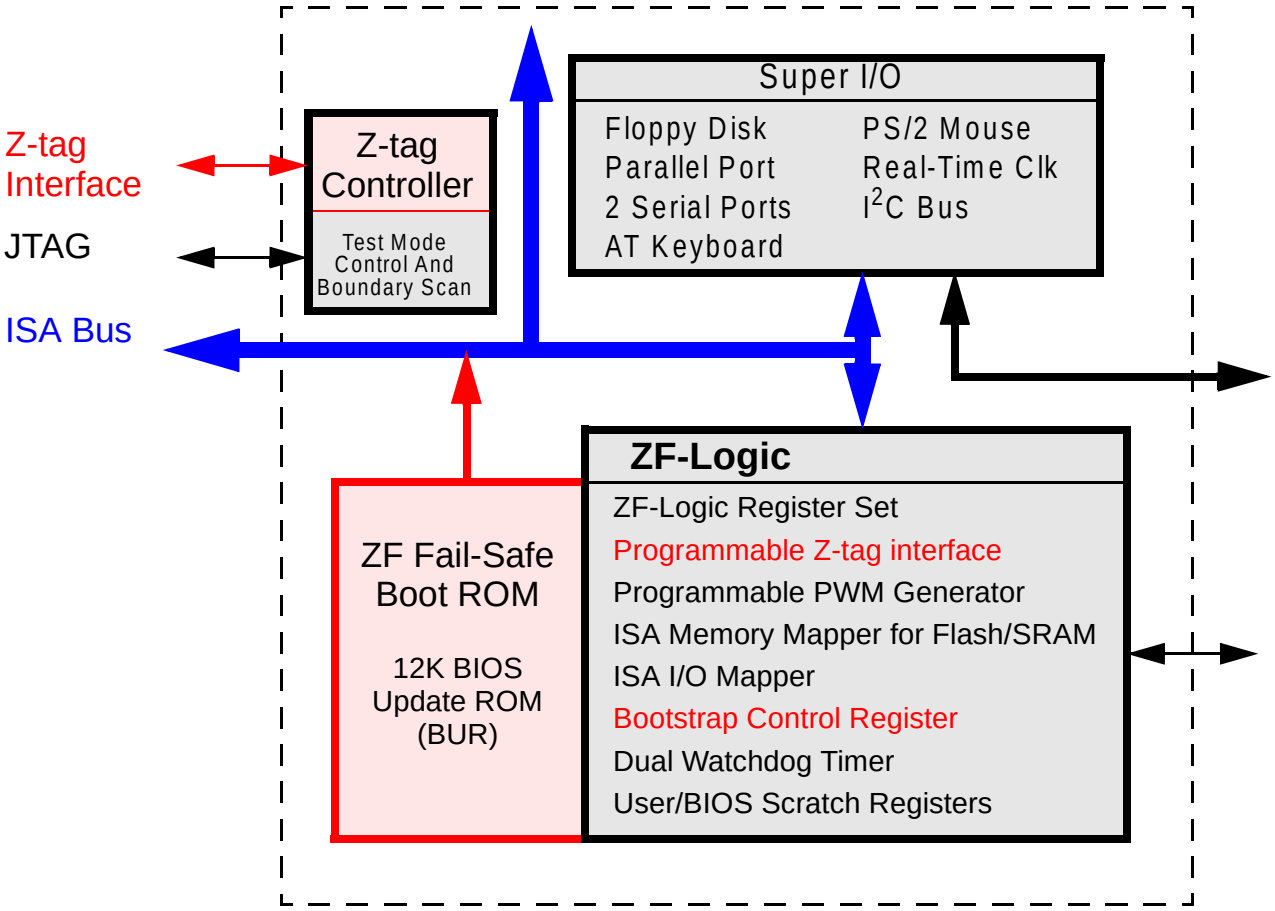
NOTE: See 'Z-tag and BUR' on page 427 in the MachZ Data Book

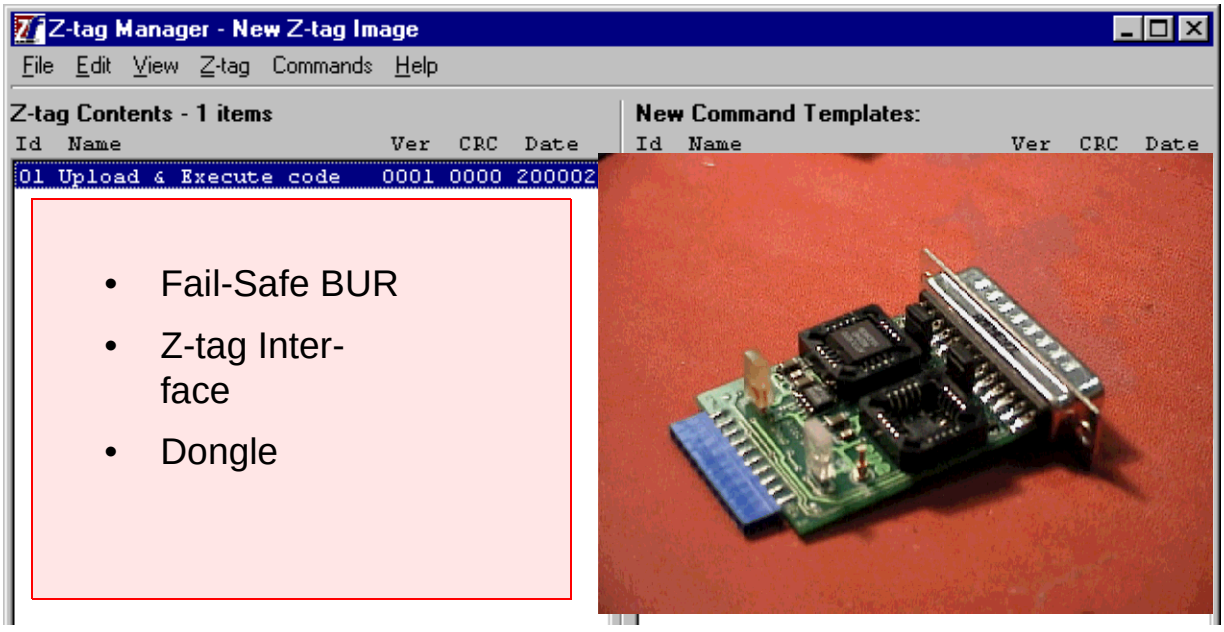
NOTE References compatible with MachZ Data Book version 0.752.

44 MachZ Block Diagram



45 Z-tag Components



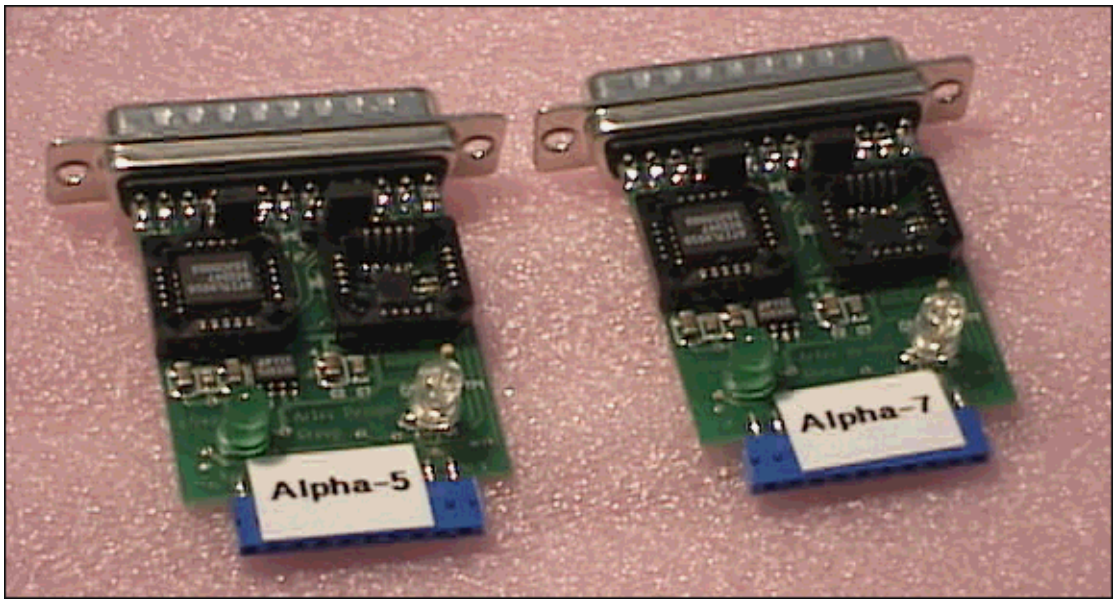


46 Z-tag Overview - Flash Update Features

- Improves speed over using serial interface.
- Frees legacy ports from system FLASH update function
- Creates a dedicated and simple interface for system upgrading.
- Advantages
    - Always Present
    - Fast
    - Up to 1.2 Mbits/S
    - Simple to use
    - Automatic
    - Easy Configuration

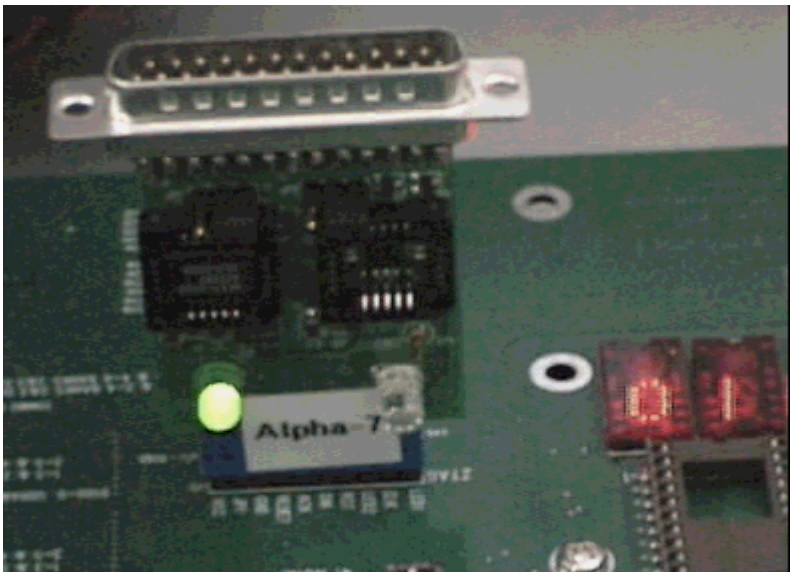
47 Z-Tag Uses

- Initial BIOS Load
- Manufacturing Test
- Diagnostics
- Remote Console
- BIOS Updates
- Application Patches

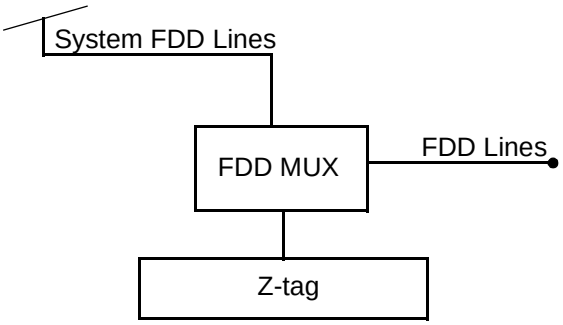


48 Z-tag Interface Mechanical Connection

- Small size
  - 14 Pins: 7x2 Dual Row Header
  - Can use Surface Mount or Through-Hole
- Placement
  - Anywhere that is convenient
  - Ideally at edge of board
    If access is possible by means of a door or panel, use is simplified

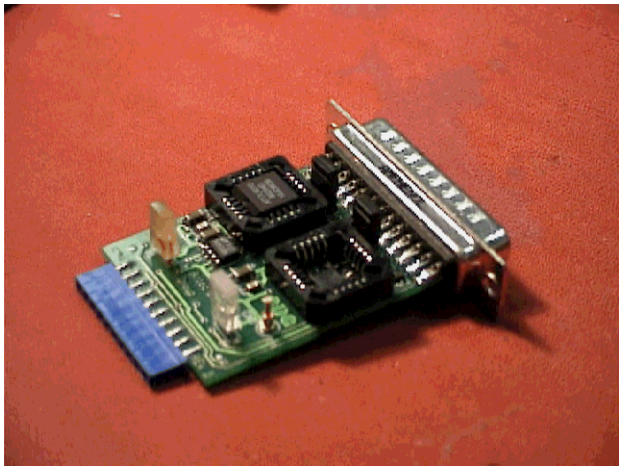


NOTE: Shared pins with floppy interface to eliminate the use of additional pins on the MachZ



49 Z-tag Dongle

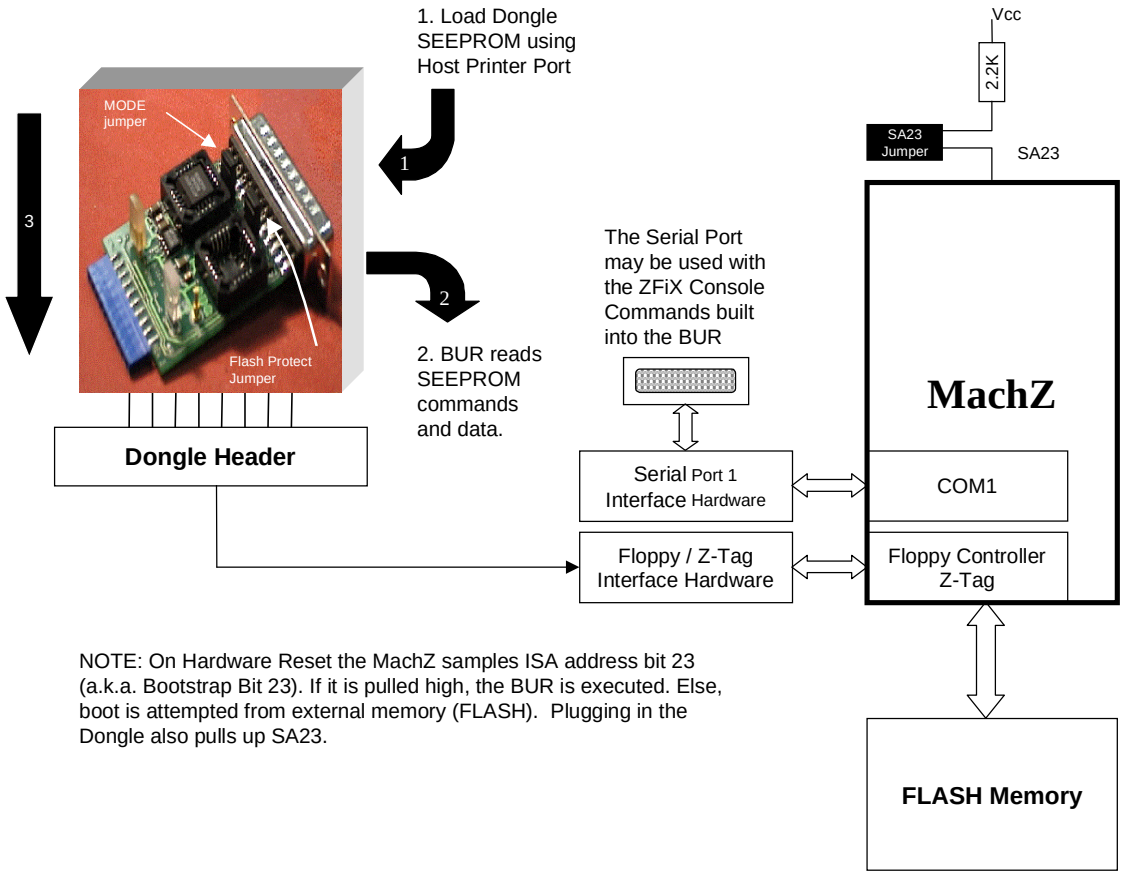
- Simple inexpensive device.
- Facilitates field upgrades
- Can store up to 512KB of data
- Provides positive feedback to operator



- 2 SEEPROMS
  - Each can be:
  - 128 or 256 KBytes
- 2 Jumpers
  - Normal / Pass Through
  - Write Protect
- 2 LEDs
  - Program Status

Name	Location	Function	Comments
Mode	Dongle	Pass-Through or Normal	Normal Connects CLK to ACK when Dongle Plugged In
Write Protect	Dongle	Protect SEEPROM(s)	
SA23	Host Board	Cause BUR Boot	Automatically set by Dongle -- add DIP switch to provide BUR boot w/o Dongle
CLK to ACK	Host Board	Used to Read Host Board SEEPROM	Not Needed if Dongle SEEPROM providing Data

Normal vs. Pass Through Mode



- 1-2: Normal MODE: Load Dongle with Host, Carry to Target
- 3. PASS THROUGH MODE: Host Connected to Target directly (through dongle)

1-2: Normal MODE: This is a two-step operation: (1). The Dongle is plugged into a host PC to load the Dongle SEEPROM. (2) The Dongle is plugged into the MachZ board. The BUR reads the Dongle SEEPROM. CLK and ACK direct connect. Data comes from the on-Dongle SEEPROM.

1. Dongle attached to Host. Use the Z-tag Manager Windows application to download commands and code images into the Dongle SEEPROM.

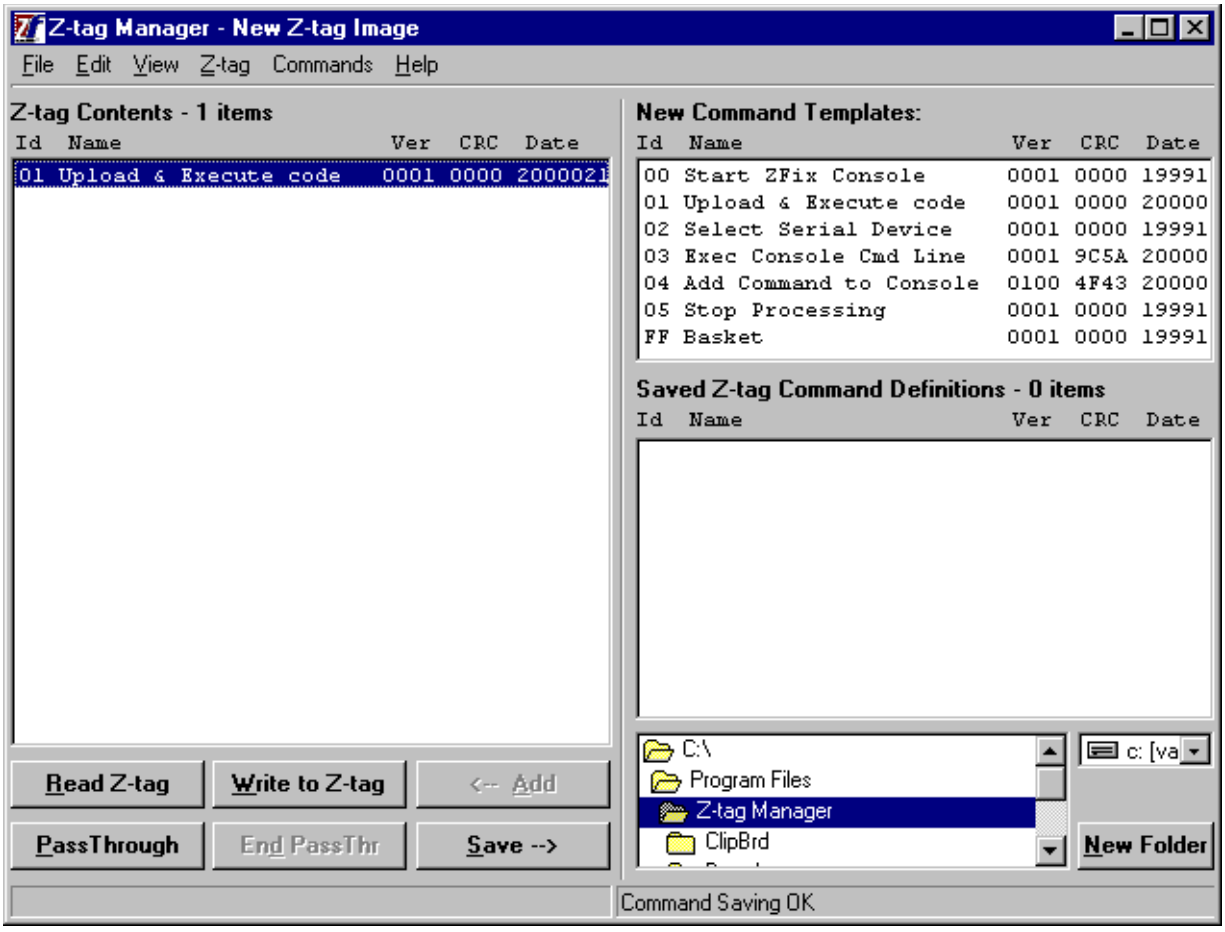
2. Dongle is attached to header pins. On reset the BUR will read commands from the Dongle SEEPROM. These commands can include an FLASH image as a data “packet”. With the MODE jumper in the “normal” mode, the CLK signal from the MachZ is directly fed back to the MachZ as the ACK signal, allowing maximum speed transfers.

3. PASS THROUGH MODE: The Mode jumper is set to Pass-Through. A printer cable from the host uses the Dongle as a Pass-Through adapter to the Dongle Header. A Z-tag Manager Software Command Sequence is executed directly from the HOST upon reset/boot of the MachZ.

ACK and Data are provided by the Host in response to CLK.

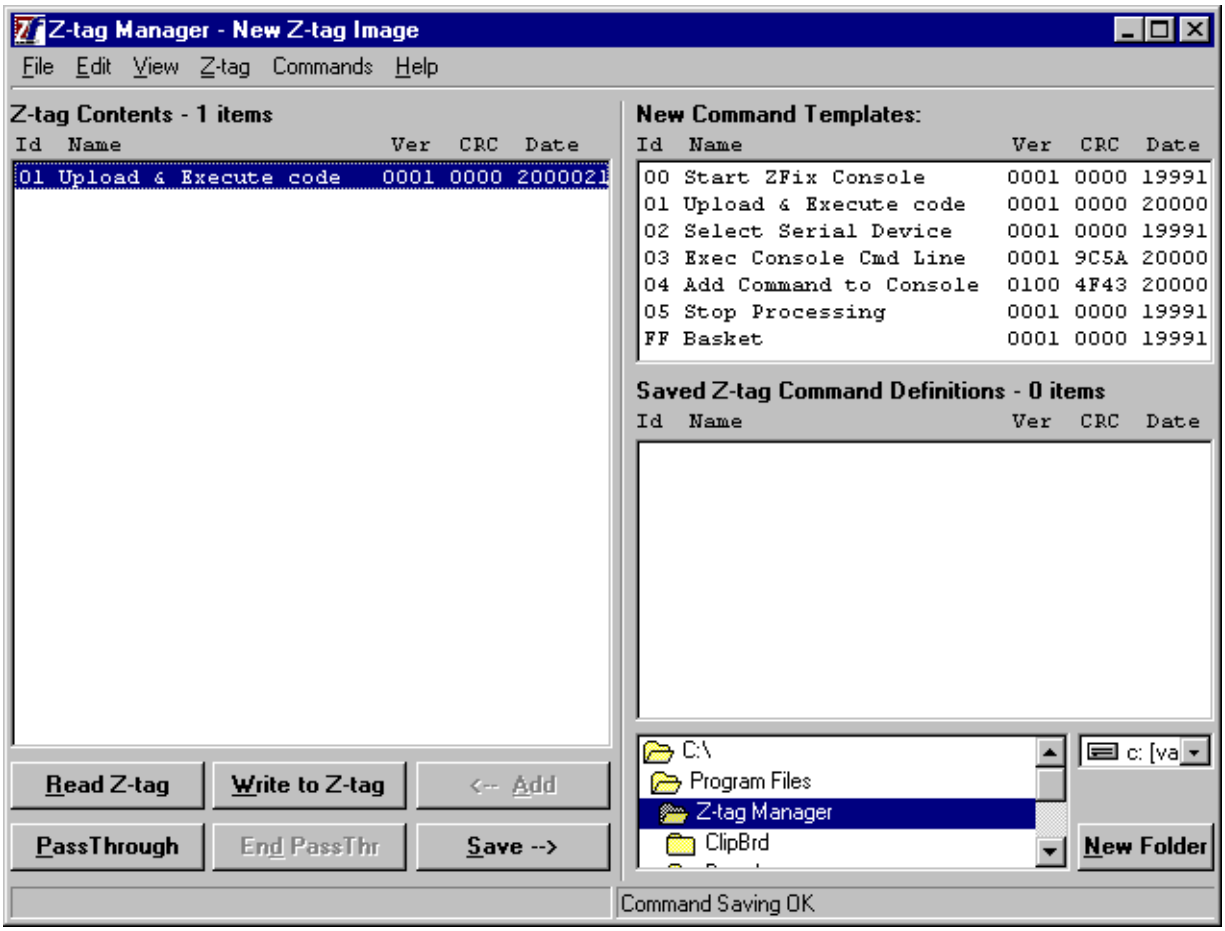
51 Z-tag Manager

- MS Windows Application
- Used to Pre-Load the “Dongle” with Data
- Simple Instructions / Powerful Results
- Only 7 Standard Commands
- The User can create their own Commands to add functionality for special purposes

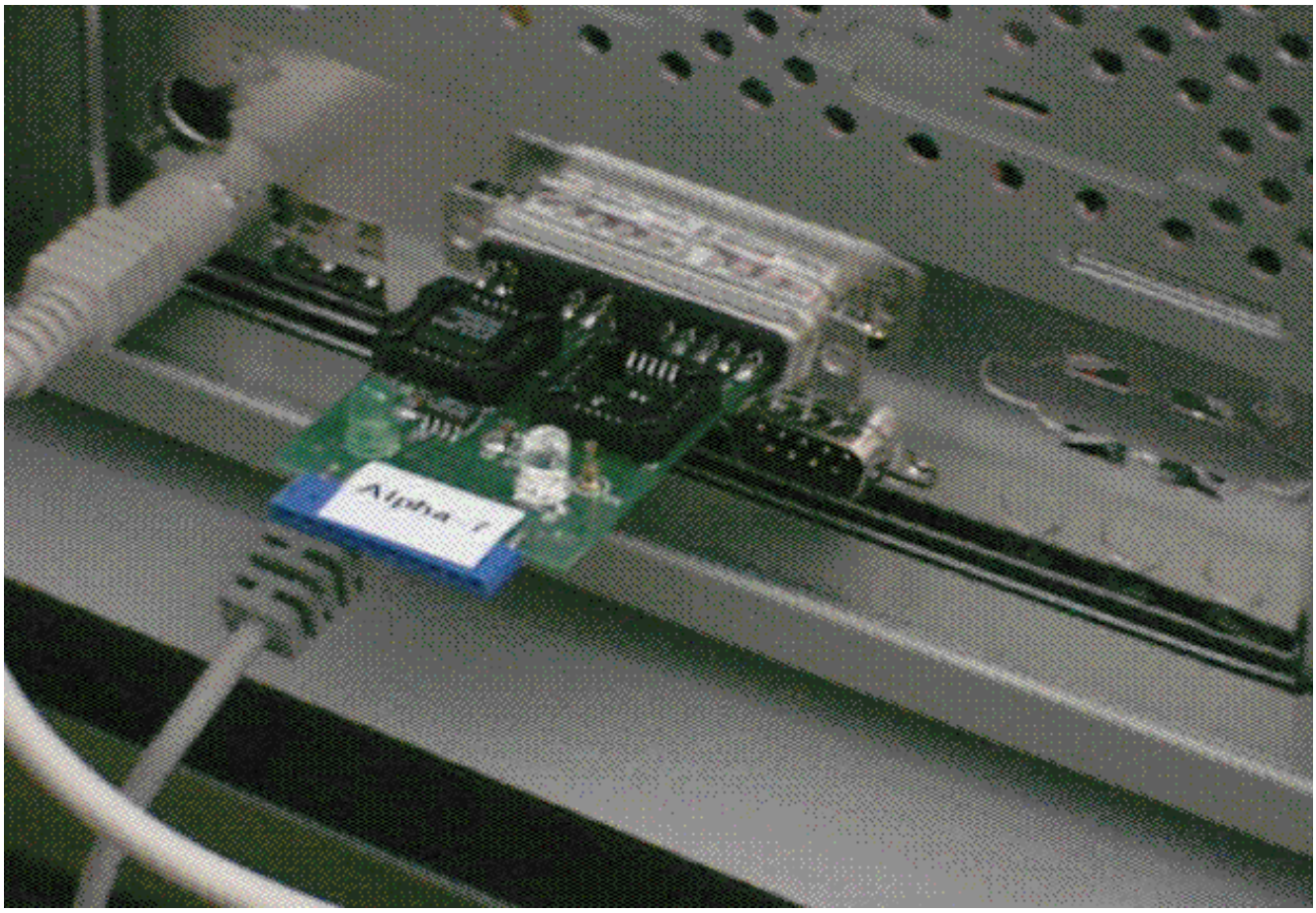


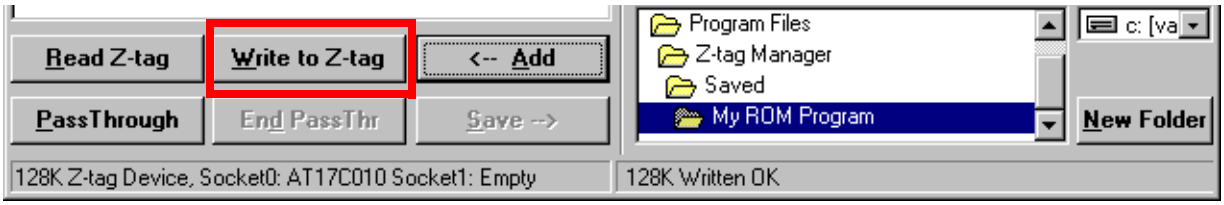
Z-tag Manager Commands

- **Dongle Related**
  - 01 - Upload and Execute Code
  - 05 - Stop
  - FF - (Transport Container "Basket" for Data)
- **Console Related**
  - 00 - Start/Resume BUR Console
  - 02 - Serial Console Mode (toggle)
  - 03 - Execute Console Command Line
  - 04 - Add Command to Console



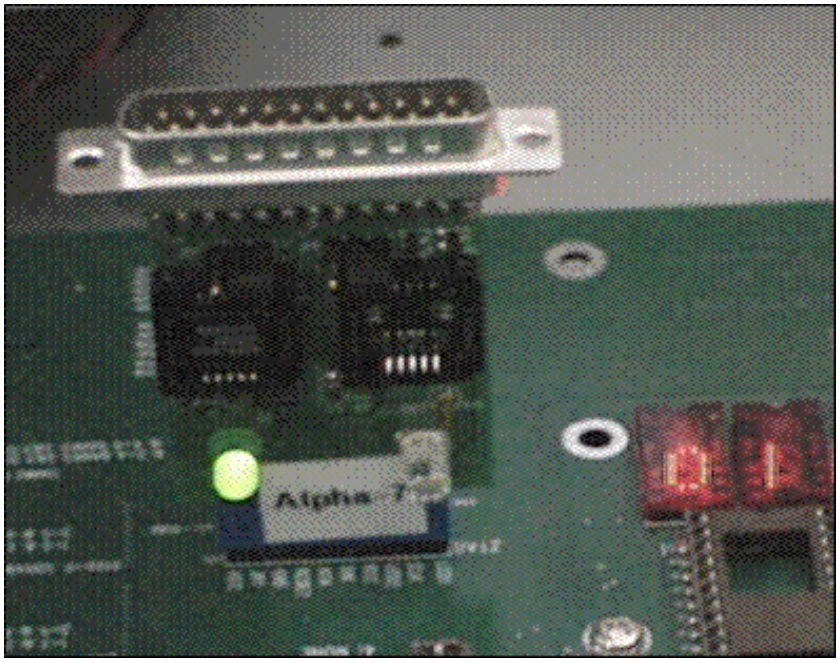
Z-tag Programming



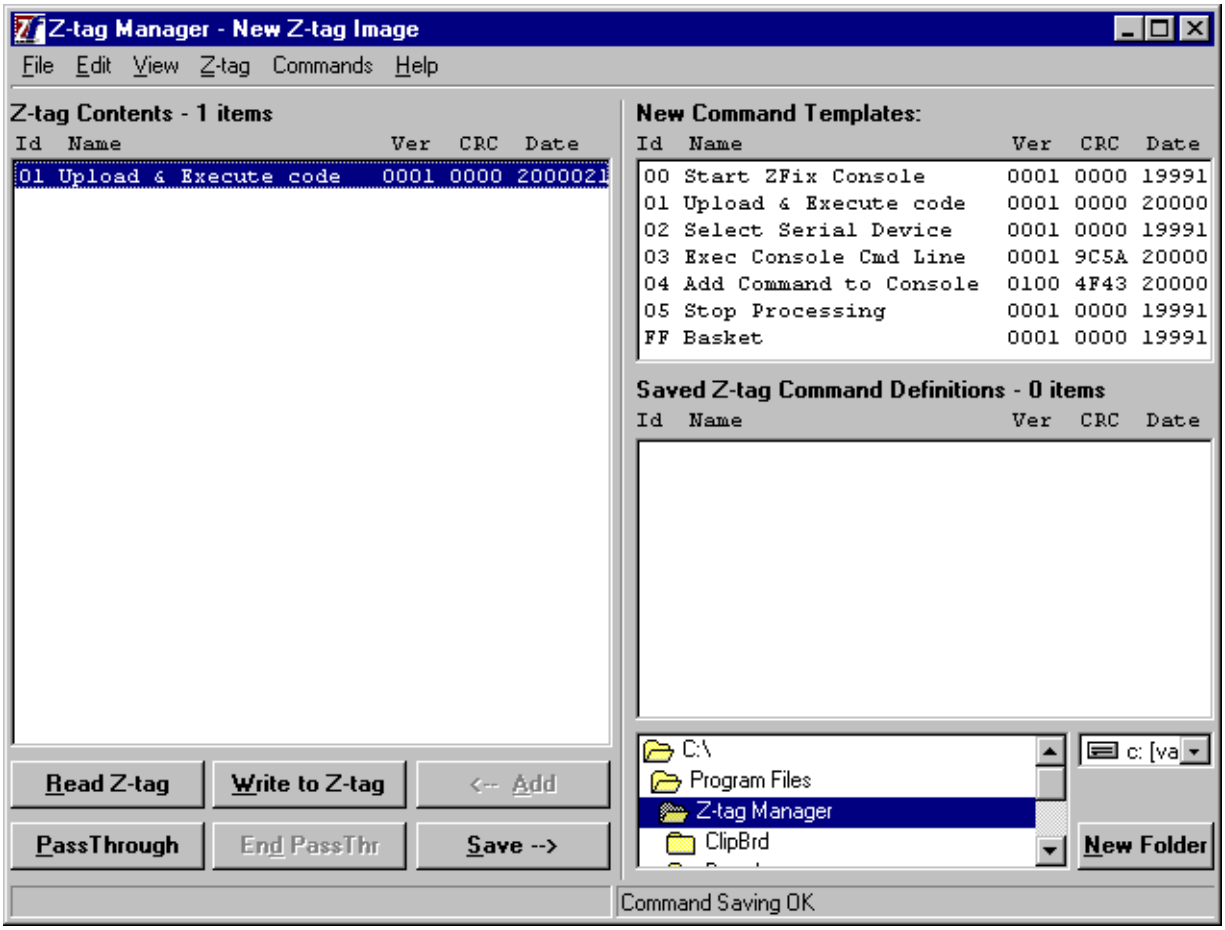


programming the dongle

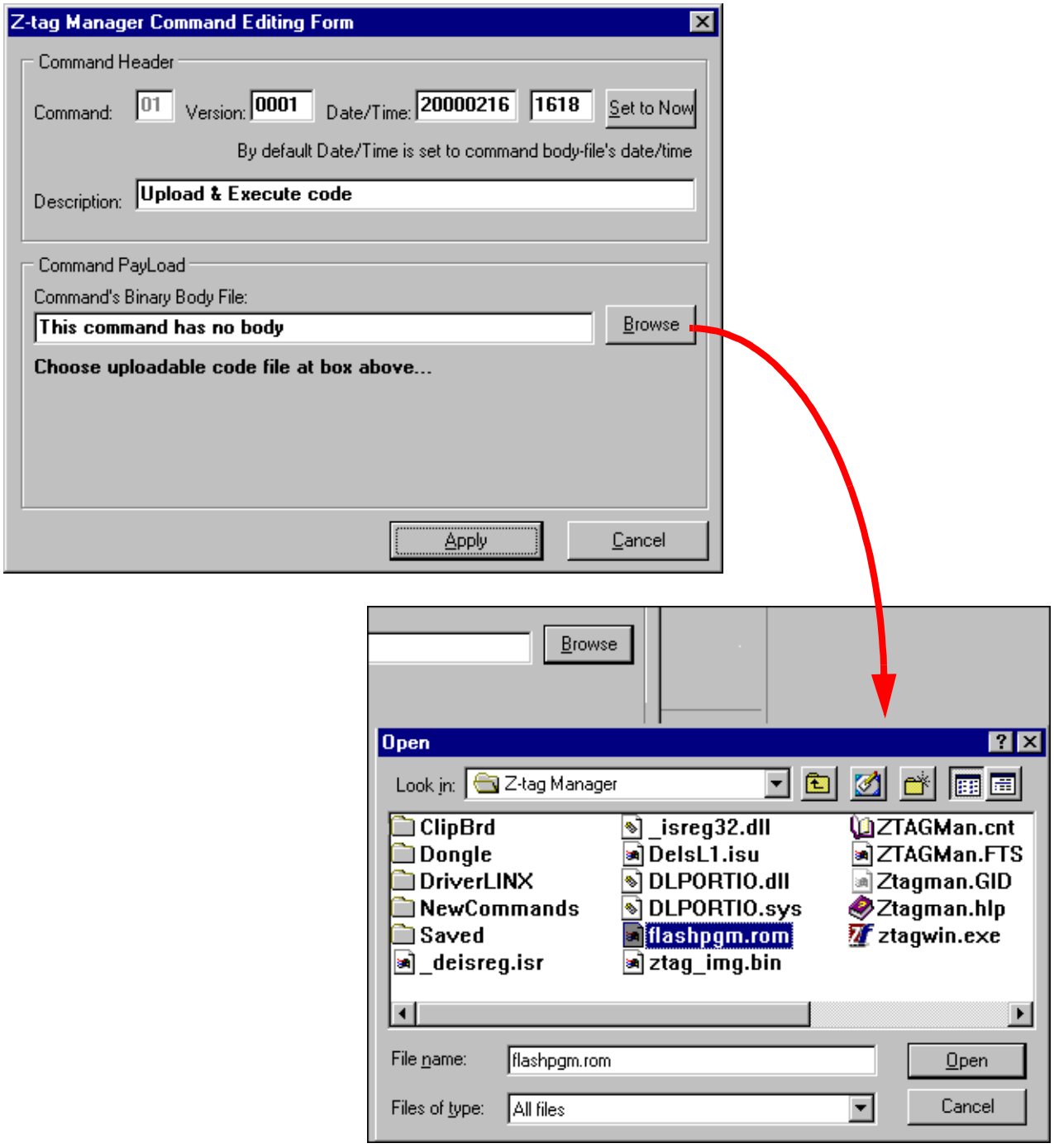
Z-tag BIOS Update



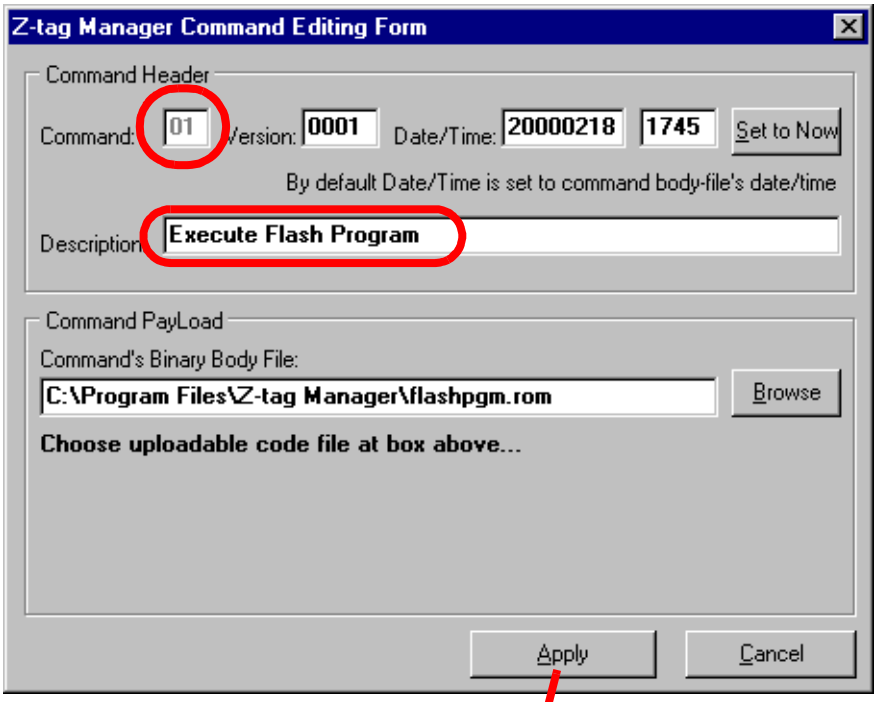
Put BIOS in Dongle - First Get Flash Pro-



56 Editing Command 01 - Upload and Execute

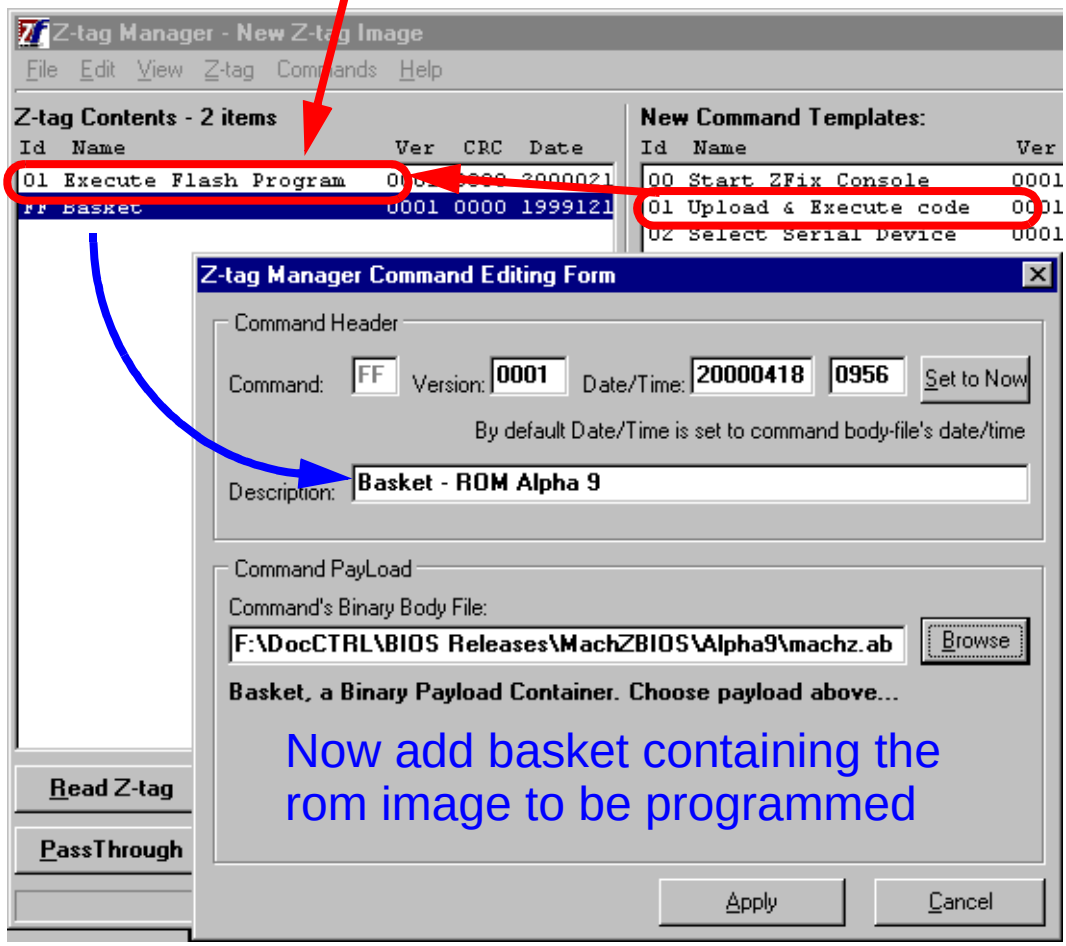


Edit Selection of Flash Code, Add BIOS Bas-

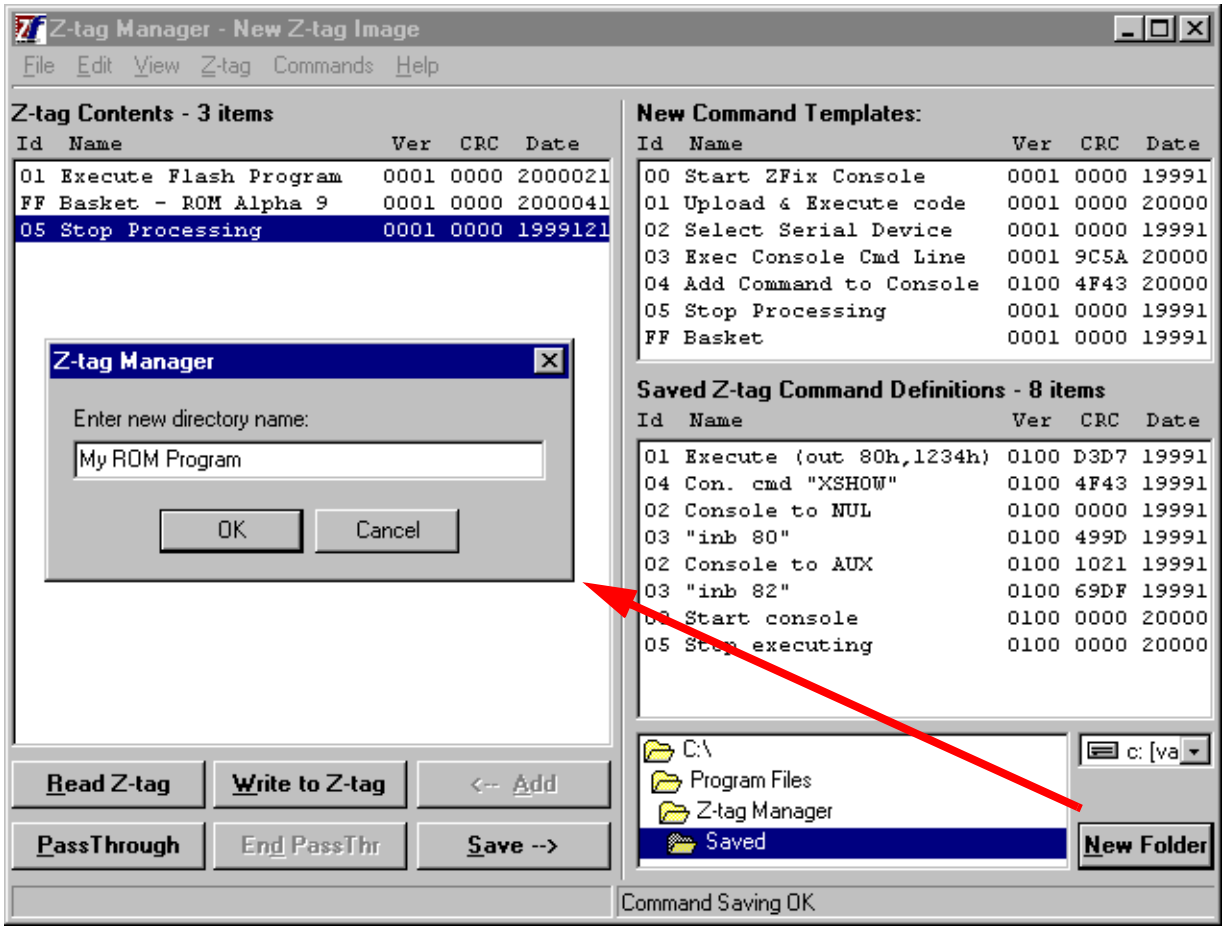


changed the name of command 01

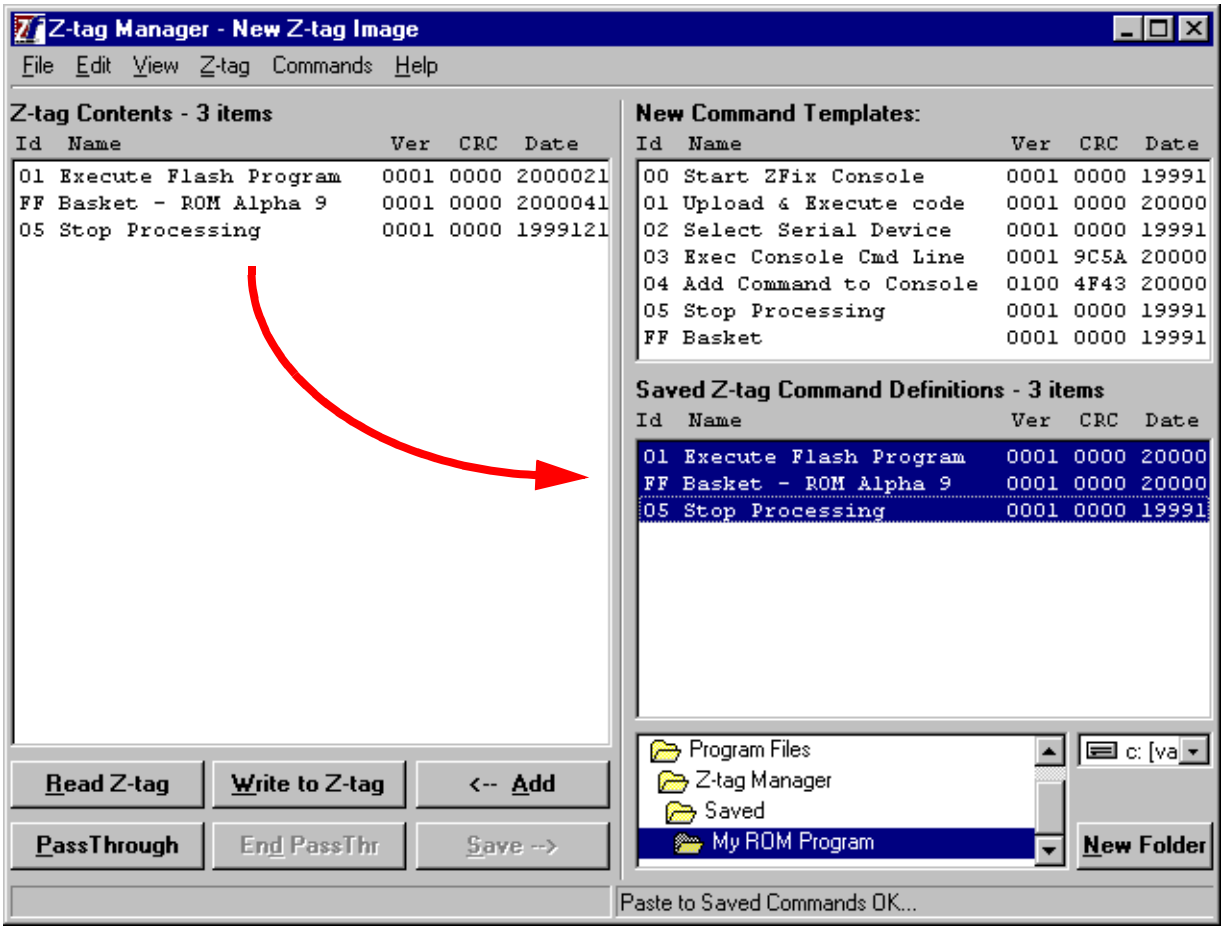
specified the file which contains the code to program the flash



58 Add Stop Command, Create Folder to Save

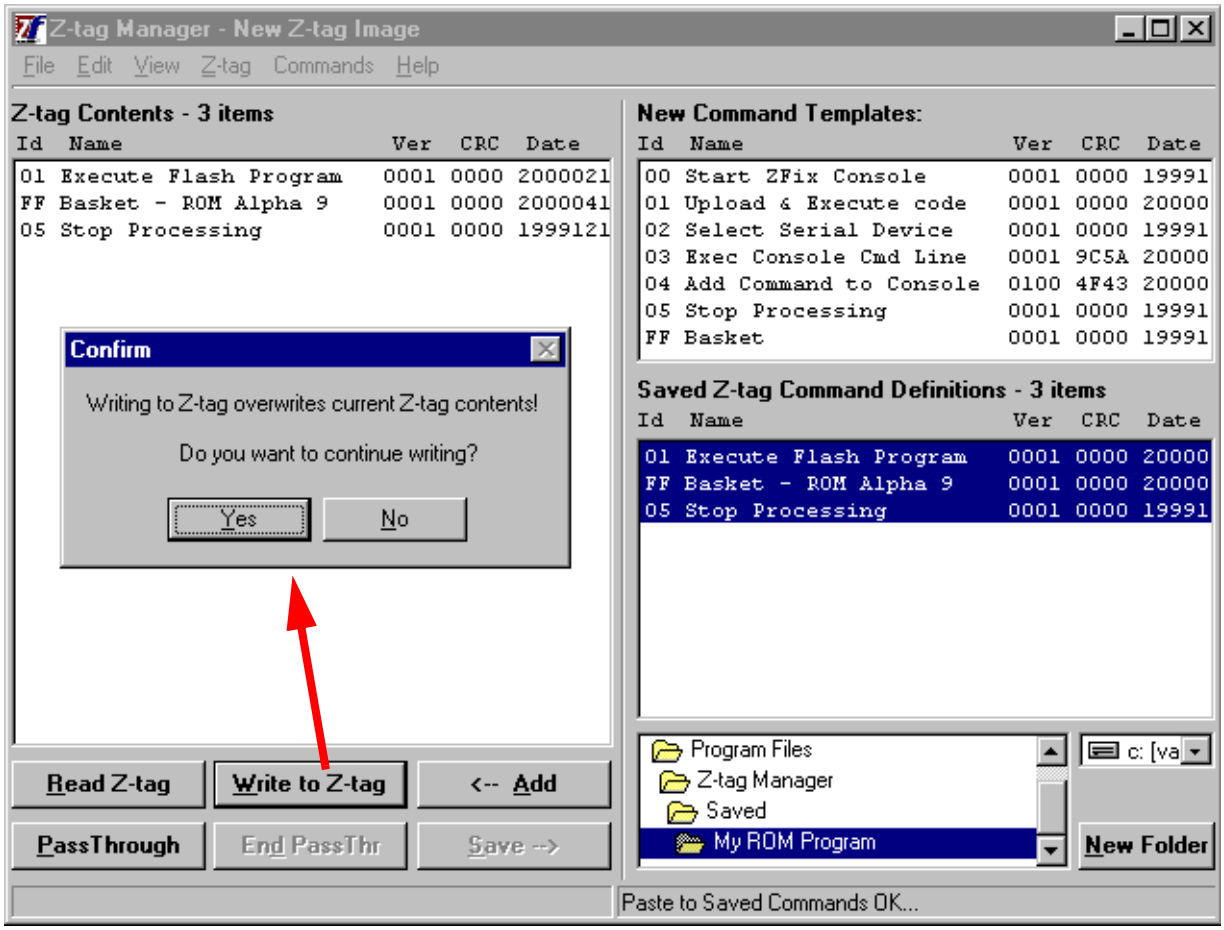


59 Copy and Paste Commands to Work Area

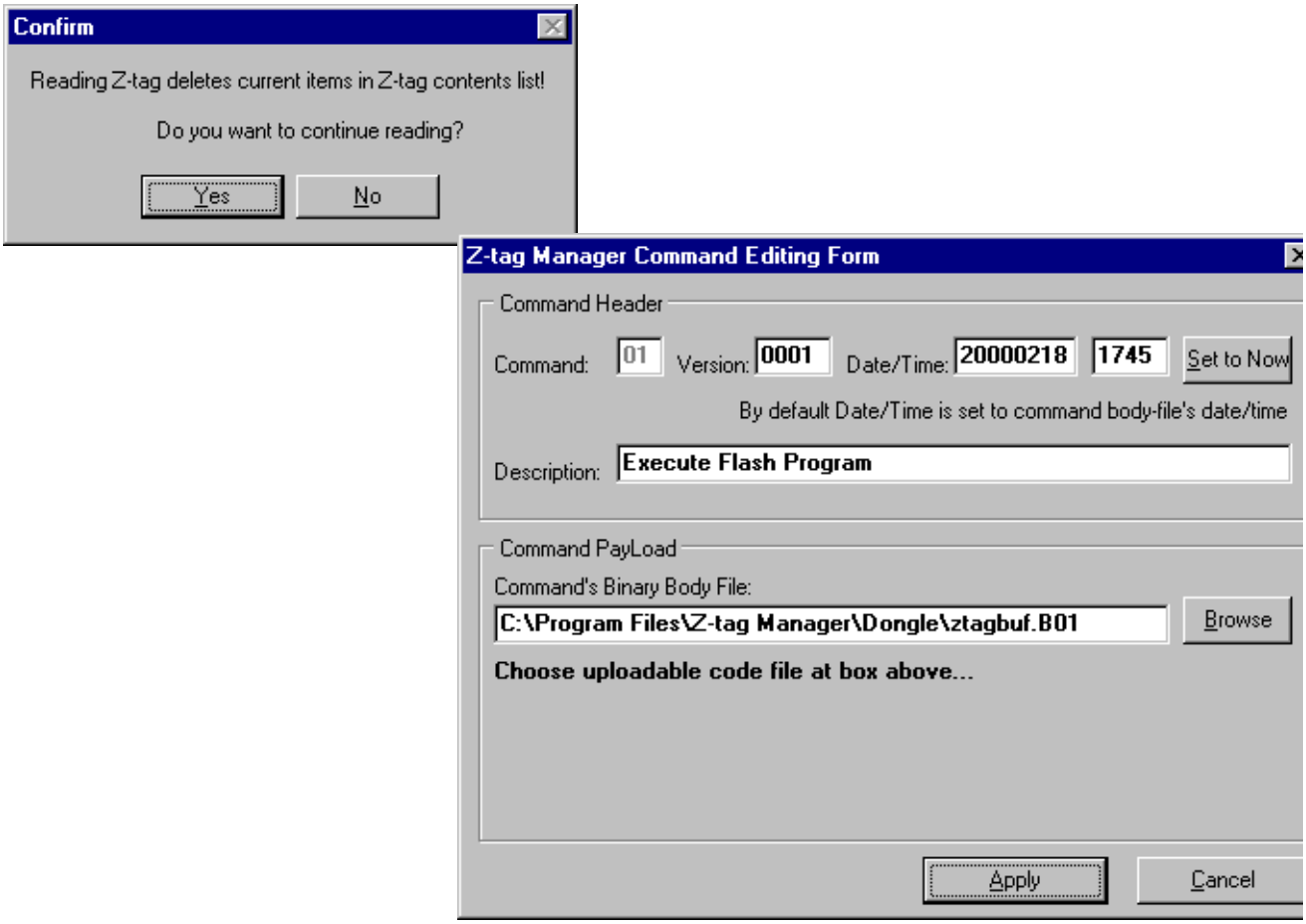


When you paste into the Saved Z-tag Command Definitions, the Z-tag manager saves it in the current folder.

60 Copy our Program to the Dongle



61 Test by Reading Back from the Dongle



The program is stored in an allocated buffer when you read in, not over your saved command.

62 BUR Version Test Program Source Code

```
;; Copyright 2000 ZF Embedded, Inc. All rights reserved.
title MachZ BET Code sample Obtains BUR Version Number

; build statements:
; ml /Fl burver1.asm
; exe2com burver1 0  (this routine available on ZF web site)

.486
burrom          segment USE16 at 0f000h
; Services table. These are the function pointers for
; uploaded code use.
                org     0ff00h  ; f000:ff00 in BUR ROM
Bur_Version     db      ?

                org     0ff0ah
CRLF            label   far     ; call ff00:ff0a --> CR/LF to COM1
                org     0ff22h
SerOut16        label   far     ; call ff00:ff22 --> AX to COM1 as decimal
                org     0ff3ah
SerSend         label   far     ; call ff00:ff3a --> Charout to COM1
burrom          ends

CODE            segment USE16 'CODE'
                assume cs:code

START:
                push    cs
                pop     es
                mov     di,offset VerText       ; ES:DI - text to show
                xor     cx,cx                   ; display until 0 reached
                call    SerSend

                les     bx,psBur_Version        ; es:bx --> Bur_Version String
                mov     ax,es:[bx]

                call    SerOut16                ; display AX to COM1 as decimal
                call    CRLF                    ; CR/LF to COM1

                retf                            ; resume with BUR

psBur_Version   dd      Bur_Version             ; define string pointer
VerText         db      'BUR Version: ',0

CODE            ends
                end     START
```

63 Ztag “Reviews”

- Try it… You’ll Like It !
- Once you switch, You’ll Never go Back.
- “The best thing since sliced bread”(Except for the MachZ itself)

64 BUR (Fail-safe) Boot Up ROM

- BUR is built-in software that serves as a prototype debug tool and Flash update utility. BUR is an internal 12K binary ROM image.
- Functionality can be divided into four categories:
    - Basic component initialization
    - Elementary debugger console functionality through COM1
    - Data fetch and execution through Z-tag interface
    - Basic OS functionality for user code
- Uses of the BUR
    - Manufacturing/field tool
    - Debug tool
    - Fail-safe System

65 Basic Component Initialization

After initialization, following system components are active:

- North Bridge
- South Bridge
- ISA Bus
- Internal Static RAM
- IRQ controller
- Timer (8259)
- COM1
- Z-tag interface

ZFix Console Commands

Command	Action
i[n[b]]/inw/ind <port>	read 8/16/32-bit value from port
o[ut[b]]/outw/outd <port> <value>	write 8/16/32-bit value to port
zfr <register>	read 8-bit value from ZFLogic register
zfw <register> <value>	write 8-bit value to ZFLogic register
db/dw/dd <address>	display memory in byte/word/dword mode
d	display next memory page in previous mode poke[b]/ pokew/poked <address> <value(s)> -
linear	use linear mode addressing
real	use real mode addressing
h[elp]/?	show help
ver	display verson information
speed <96/19/38/56/115> <hs>	serial speed. Set hs to 1 for RTS/CTS[a]
yload <address>	load data through YModem to address
ysend <address> <length> [filename]	send data through YModem from address
g[o] <address>	start executing from address
dls	Display available download segment address

a.The default speed on power up is 9600. The <hs> handshake bit is currently not working. You may try higher speeds, but you may lose data.

```
ZFiX - MachZ PCe Internal Console
(c)1999 ZF Embedded, Inc.
: DLS
 0070                                   // Display available segment for downloads
: yload 70:0
Please start YModem transmission now or press <ESC> ...
CCCCCCCCCCCC  ← waiting for transfer
YModem data transfer succeeded.
: g 70:0
BUR Version: 0101
```

67 Data Fetch and Execute

There are 7 different type of commands which the BUR understands and executes:

- 00 - Start/Resume BUR console.
- 01 - Upload and execute code.
- 02 - Serial console mode.
- 03 - Execute console command line.
- 04 - Add command to a console.
- 05 - Stop.
- FF - Basket

```
;; Copyright 2000 ZF Embedded, Inc. All rights reserved.
title MachZ BET Code sample Obtains BUR Version Number

; build statements:
; ml /Fl burver1.asm
; exe2com burver1 0  (this routine available on ZF web site)

.486
burrom          segment USE16 at 0f000h
; Services table. These are the function pointers for
; uploaded code use.
                org     0ff00h  ; f000:ff00 in BUR ROM
Bur_Version     db      ?

                org     0ff0ah
CRLF            label   far     ; call ff00:ff0a --> CR/LF to COM1
                org     0ff22h
SerOut16        label   far     ; call ff00:ff22 --> AX to COM1 as decimal
                org     0ff3ah
SerSend         label   far     ; call ff00:ff3a --> Charout to COM1
burrom          ends

CODE            segment USE16 'CODE'
                assume cs:code

START:
                push    cs
                pop     es
                mov     di,offset VerText       ; ES:DI - text to show
                xor     cx,cx                   ; display until 0 reached
                call    SerSend

                les     bx,psBur_Version        ; es:bx --> Bur_Version
String
                mov     ax,es:[bx]

                call    SerOut16                ; display AX to COM1 as
decimal
                call    CRLF                    ; CR/LF to COM1

                retf                            ; resume with BUR

psBur_Version   dd      Bur_Version             ; define string pointer
VerText         db      'BUR Version: ',0

CODE            ends
                end     START
```

69 Manufacturing and Field Tool

- Manufacturing cycle can use the dongle at two stages:
  - 1st load the Manufacturing Test program to allow diagnostic exercising of the device.
  - Last operation to update the device as it goes out the door.
- Very inexpensive device to provide to the field personnel to trouble-shoot on-site.

BUR: Debug tool

- During initial bring-up the BUR allows designer to not populate any other devices on the board with exception of the power and clock circuits.
- MachZ DRAM configurations were all debugged using the BUR code only.
- Small applications can be brought in to the device and always executed from Z-tag interface

71 Fail-safe System

- Uses Z-tag interface, dual WDT and BUR code to provide automatic recovery of the system.
- One recovery scenario:
- Application overwrites a portion of the OS with Watch Dog enabled.
- Application hangs due to lack of OS.
- WDT re-boots into BUR code (bit 23 tied high)
- Bur code boots, finds Z-tag device on port and loads through the port the information present in the serial EPROM on board.
- The code present in the serial EPROM includes the CRC for the flash device. The BUR code then calculates the flash CRC and compares it to the expected value.
- If CRC is bad, additional programs can be downloaded through the Z-tag interface to execute a modem call to the factory requesting new code.
- Once new code is downloaded, the flash is updated and a system reset is executed (a South Bridge Reset or a jump to ffff fff0) after setting the BUR base address to zero in index register 57H/58H.

ZF Linux Devices, Inc.

1052 Elwell Court

Palo Alto, California 94303

(650) 965-3800 · Fax 965-4050

www.zfmicro.com